Radboud Repository

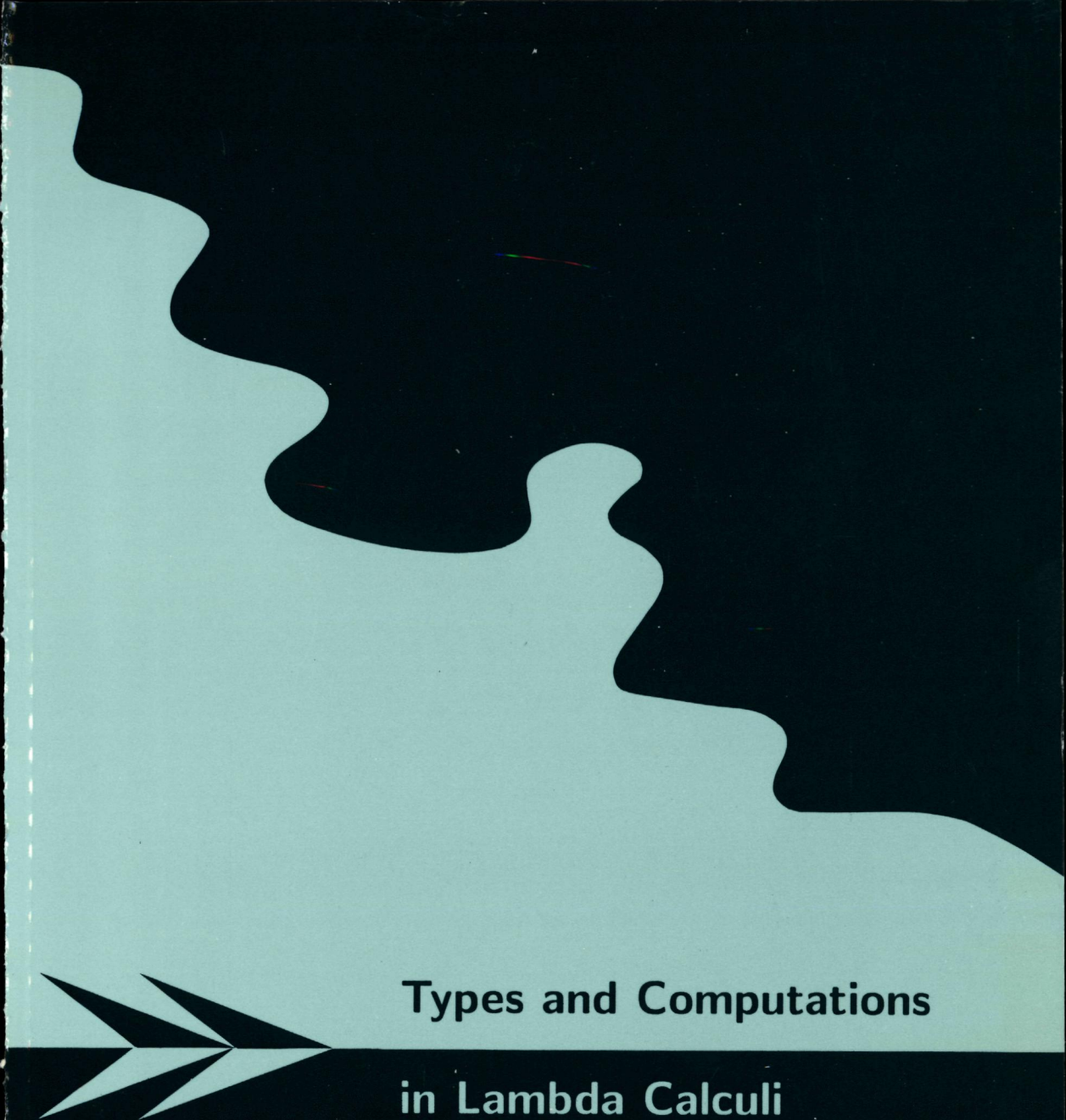
Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/145918

Please be advised that this information was generated on 2017-12-05 and may be subject to change.



Types and Computations in Lambda Calculi and Graph Rewrite Systems

Erik Barendsen

Types and Computations
in Lambda Calculi and Graph Rewrite Systems

Cover design Edith Barendsen
Print Febodruk, Enschede

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Barendsen, Erik

Types and computations in lambda calculi and graph rewrite systems / Erik Barendsen [S l s n] (Enschede Febodruk) Ill
Proefschrift Nijmegen – Met index, lit opg – Met samenvatting in het Nederlands
ISBN 90-9007964-5
Trefw lambda calculus / programmeertalen / typen (semantiek)

Types and Computations in Lambda Calculi and Graph Rewrite Systems

een wetenschappelijke proeve op het gebied van de Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen
op donderdag 2 februari 1995,
des namiddags te 3.30 uur precies

door

Erik Barendsen

geboren op 10 juni 1966 te Zutphen

Promotor: prof dr H.P. Barendregt

Co-promotor: dr. M.A. Bezem
(Universiteit Utrecht)

Voor mijn ouders

Acknowledgements

Many people have had their influence on the form and contents of this thesis

I am endebted to my promotor Henk Barendregt for creating an excellent working environment with plenty of freedom to choose my research topics His enthousiasm and knowledge of the field have been most stimulating My co-promotor Marc Bezem provided valuable support and advice

The manuscript committee consisted of Thierry Coquand, professor van Dalen and Jan Willem Klop I thank them for their judgement

Research is often teamwork, I very much enjoyed the collaboration with co-authors Marc Bezem and Sjaak Smetsers

Moreover, I would like to thank Steffen van Bakel, Thierry Coquand, Jean-Pierre van Draanen, Herman Geuvers, Marco Hollenberg, Bart Jacobs, and Jan Terlouw. for many interesting discussions and comments on earlier versions of parts of this thesis Also the contacts with the participants of the Lambda Inter-city seminar were helpful

The working atmosphere in Nijmegen was a very pleasant one, due to my collegues of the groups Foundations and Functional Programming

The diagrams in parts I and III were produced using Paul Taylor's diagrams package for TEX The pictures in Part II were all drawn by Sjaak Smetsers Thanks to my sister Edith for taking care of the cover

The research reported here has been supported by the Netherlands Computer Science Research Foundation (SION) with financial support of the Netherlands Organization for Scientific Research (NWO)

Contents

1 Introduction and Overview 1
Aspects of Typing 1
Part I Lambda Calculus 3
Part II Graph Rewrite Systems 5
Part III Numerations 9

I Lambda Calculus

2 Lambda Calculus with Higher-Type Oracles **13**
2 1 Syntax 13
2 2 The Church-Rosser Property 17

3 First- and Second-Order Lambda Calculus **29**
3 1 Syntax 29
3 2 Semantics of First-Order Systems 34
3 3 Semantics of Second-Order Systems 37
3 4 Building Per Structures over Type Structures 45
3 5 Interpretations in Extended Type Structures 50
3 6 A Full Per Model 51
3 7 Kleene Recursion 54

4 Bar Recursion versus Polymorphism **57**
4 1 Introduction 57
4 2 Syntax 58
4 3 Semantics 64
4 4 Realizability of Functional Specifications 66
4 5 First-Order Non Conservativity 69
4 6 Non-Realizability in Second-Order Lambda Calculus 74
4 7 Extensions to Higher-Order Lambda Calculus 76

II Graph Rewrite Systems

5 Conventional Typing in Graph Rewrite Systems **81**
5 1 Introduction 81
5 2 Graph Rewriting 82
5 3 Conventional Typing 90

6 Uniqueness Typing in Graph Rewrite Systems **101**
6 1 Introduction 101
6 2 Informal Explanation 103
6 3 Usage Analysis 107
6 4 Uniqueness Types 111
6 5 Type Assignment 117
6 6 Uniqueness Type Environments 119
6 7 Typing Redexes 126
6 8 Locality Properties of Primary Redexes 129
6 9 Saturated Markings 131
6 10 Marking Reducts 136
6 11 Correctness of Reduct Markings 140
6 12 Typing Reducts 152
6 13 Applications 155
6 14 Future Research 155

III Numerations

7 Notes on Fixed Points **159**
7 1 Introduction 159
7 2 Examples 159
7 3 Numerations 161
7 4 Code Dependent Fixed Point Results 164
7 5 Fixed Points of Endomorphisms 166
7 6 Effective Versions of the Recursion Theorems 174

Bibliography **177**

Index **183**

Samenvatting **185**

Curriculum Vitae **187**

Chapter 1

Introduction and Overview

Aspects of Typing

The central subject of this thesis is *typing*. In typing systems, types are assigned to objects (in some context). This yields to *typing judgements* of the form $a \quad \tau$, where a is an object and τ a type. Traditionally, there are two views on typing: the *computational* and the *logical* one. They differ in the interpretation of typing judgements.

In the computational setting, a is regarded as a *program* or *algorithm*, and τ as a *specification* of a (in some formalism). This gives a notion of partial correctness, namely well-typedness of programs.

From the logical point of view, τ is a *formula*, and a is considered to be a *proof* of τ. Various logical systems have a type theoretic interpretation. This branch of type theory is related to proof theory because the constructive c.q. algorithmic content of proofs plays an important role. The result is a realizability interpretation: provability in the logical systems implies the existence of an 'inhabitant' in the corresponding type system. For more information on this logical interpretation, the reader is referred to Barendregt (1992), see also Geuvers (1993) for an extensive overview of variants of this propositions-as-types concept.

In this thesis we focus on the computational aspect of type theory. The objects are either λ-terms or graphs.

Church versus Curry Typing

One can distinguish two different styles in typing, usually called after their inventors A. Church and H. B. Curry.

The approach *a la* Church assembles typed objects from typed components, i.e. the objects are constructed together with their types. The atomic objects are typed variables and typed constants (for basic, predefined operations). The typed λ-calculi in this thesis are Church style. We use type annotations to indicate the types of basic objects: if $\mathbf{0}$ denotes the type of natural numbers, then $\lambda x^{\mathbf{0}}\, x^{\mathbf{0}}$

denotes the identity function on $\mathbb{N}$, having type $\mathbf{0}\to\mathbf{0}$. Each typed term has exactly one type

In the approach *à la* Curry, types are assigned to existing untyped objects, using *typing rules* that refer to the structure of the objects in question. In these systems, the typability problem (given an object, does it have a type?) becomes interesting. In the Curry systems, it is possible that an object has more than one type. The identity function $\lambda x.x$, for example, has type $\mathbf{0}\to\mathbf{0}$, but also type $(\mathbf{0}\to\mathbf{0})\to(\mathbf{0}\to\mathbf{0})$. In the case of graph rewrite systems, we only work in Curry-style systems.

In type systems with both a Church and a Curry version (notably λ-calculi) one usually has the following correspondence: If M is an annotated λ-term, then

$$M : \tau \quad (\grave{a}\ la \text{ Church}) \quad \Rightarrow \quad |M| : \tau \quad (\grave{a}\ la \text{ Curry}),$$

where $|M|$ denotes the untyped λ-term obtained from M by erasing all type annotations.

The computational behaviour of (typed) terms and graphs is expressed either by a formal system for derivations of equalities (like in our λ-calculi, where

$$(\lambda x.M)N = M[x := N] \qquad (\beta)$$

is the main computational rule), or by a reduction relation (especially in our treatment of graph rewrite systems). The derivation systems can easily be extended with rules for, e.g., arithmetic and propositional connectives. Typical research questions concern the computational or proof theoretical strength of the theory, and (in the case of Curry typing) the preservation of typing during computation.

Intensional versus Extensional Typing

It is interesting to consider yet another method for assigning types to untyped objects. For the moment, we focus on simple types built from $\mathbf{0}$ using the function type constructor $\to$.

The Curry method can be considered as an *intensional* way of typing: Types are assigned to objects according to their *syntactical form*.

Now suppose A is a collection of objects with a binary *application function* (denoted by $\cdot$). Then we can classify ('type') the elements of A according to their *applicative behaviour* by setting $a : \mathbf{0}$ if a belongs to some predefined $A_0 \subseteq A$, and $a : \tau_1\to\tau_2$ if a behaves like a function from τ_1 to τ_2:

$$a : \tau_1\to\tau_2 \quad \Leftrightarrow \quad \forall b\,[b : \tau_1 \ \Rightarrow\ a \cdot b : \tau_2].$$

We refer to this as *extensional typing*. It is used to construct models of typed lambda calculus: Church-typed terms M are translated into their intensionally typed versions $|M|$, and these are interpreted as extensionally typed objects (of the same type).

Part I: Lambda Calculus

The research reported in Part I concerns syntax and semantics of certain first- and second-order λ-calculi. The first-order systems are extensions of the *simply typed* lambda calculus $\boldsymbol{\lambda}^{\tau}$ of Church (1940), with basic type $\mathbf{0}$ and type constructor $\rightarrow$.

The system of *primitive recursive functionals* $\boldsymbol{\lambda}\mathbf{T}$ results by adding constants for (higher order) primitive recursion over the algebra of natural numbers (generated using constants $\boldsymbol{0}$ and $\boldsymbol{S}$). Gödel (1958) used this system for his functional interpretation of arithmetic. It follows that a function is representable in $\boldsymbol{\lambda}\mathbf{T}$ precisely if it is provably total in first-order arithmetic.

Spector (1962) extended Gödel's interpretation to analysis by adding to $\boldsymbol{\lambda}\mathbf{T}$ a computation mechanism called *bar recursion* for recursion on well-founded trees of functional sequences. In the resulting system $\boldsymbol{\lambda}\mathbf{TB}$ one can represent exactly those functions that are provably total in analysis.

The second-order systems are variants of the *polymorphic* λ-calculus $\boldsymbol{\lambda}\mathbf{2}$ due to Girard (1972) and Reynolds (1974). In fact, Girard (1972) extended Gödel's results to second- and higher-order arithmetic, via the second-order extension $\boldsymbol{\lambda}\mathbf{2T}$ of $\boldsymbol{\lambda}\mathbf{T}$.

Polymorphism involves the internalization of *genericity*: rather than stating, for example,

$$\lambda x^{\tau}.x^{\tau} \ : \ \tau \rightarrow \tau \quad \text{for all } \tau,$$

one introduces a polymorphic identity function which depends dynamically on its input type:

$$\Lambda\alpha\, \lambda x^{\alpha}.x^{\alpha} \ : \ \forall\alpha.\alpha \rightarrow \alpha.$$

Then $(\Lambda\alpha\, \lambda x^{\alpha}.x^{\alpha})\tau$ is the identity function on type τ.

One could regard $\boldsymbol{\lambda}\mathbf{2T}$ as a type-theoretical rather than as a computational extension of $\boldsymbol{\lambda}\mathbf{T}$. Surprisingly, however, the resulting system has the same computational power as Spector's system $\boldsymbol{\lambda}\mathbf{TB}$.

This equivalence with respect to definability suggests a, possibly deep, relationship between the systems $\boldsymbol{\lambda}\mathbf{2T}$ and $\boldsymbol{\lambda}\mathbf{TB}$. The research resulting in Part I has been sparked off by the question whether the two calculi differ with respect to definability in higher types.

The first problem that arises concerns the concept of definability itself. E.g., the domain of the above functions is the set of natural numbers, but for functionals it is not clear which (higher-type) objects should be regarded as inputs. For a comparison of $\boldsymbol{\lambda}\mathbf{2T}$ and $\boldsymbol{\lambda}\mathbf{TB}$ we introduce a concept of $\boldsymbol{\lambda}\mathbf{T}$-*specification* of functionals, with a notion of realizability by terms of $\boldsymbol{\lambda}\mathbf{2T}$ and $\boldsymbol{\lambda}\mathbf{TB}$ respectively. In the case of functions (i.e., functionals of type $\mathbf{1}$) this corresponds exactly to classical definability.

It turned out that there is a higher type specification (on type level $\mathbf{3}$) that is realizable in $\boldsymbol{\lambda}\mathbf{TB}$ but not in $\boldsymbol{\lambda}\mathbf{2T}$. This suggests that $\boldsymbol{\lambda}\mathbf{TB}$ is, in some sense, stronger that $\boldsymbol{\lambda}\mathbf{2T}$.

The negative proof for **λ2T** is rather involved. It uses a *counter model* containing a specific discontinuous functional. In the standard literature, however, models of second-order lambda calculus are usually based on some continuity principle, for example coherence spaces (see Girard et al. (1989)) or complete partial orderings (see Poll (1994)). Also Girard's model HEO2 (see Troelstra (1973)) exhibits an inherent continuity expressed by the Kreisel-Lacombe-Schoenfield Theorem.

We apply a general model construction (inspired by HEO2) for second order systems based on partial equivalence relations (per). This per construction is parametrized by a (partial) applicative structure $\langle A, \cdot \rangle$. Each type is interpreted as a subset of A, using the abovementioned method of extensional typing. Equality in the theory (at type τ, say) is modelled by an equivalence relation on the objects of extensional type τ. Some care is needed: each object of type $\tau_1 \to \tau_2$ that does not respect the equivalence relations on τ_1 and τ_2 (in its applicative behaviour) is removed.

For the construction of a counter model based on pers (containing the discontinuous functional in question) one is confronted with another problem: This (higher-order) functional should be encoded in a 'flat' applicative structure. A method for doing this is provided by Kleene's (1962) concept of λ-definability in higher types, using an extension of untyped λ-calculus with *oracles* originating from a model of first-order typed lambda-calculus (a so-called *type structure*).

We use the closed term model associated with the resulting reduction system as the basis of a per construction. It is interesting to remark that the fact that interpretation in this term model is sound boils down to the property that the rules for intensional typing are 'extensionally sound': any term with intensional type τ also has extensional type τ (but not necessarily vice versa).

Combining these two techniques yields a construction for extending certain first-order models to second-order ones, which is interesting in itself. For our non-realizability result we apply this construction to the structure of all functionals over $\mathbb{N}$ (the so-called *full type structure* over $\mathbb{N}$).

Comparing this model with HEO2 one could say that HEO2 interprets subrecursive terms in a recursive domain, whereas we interpret subrecursive terms in a superrecursive domain.

One of the objectives of model theory is the investigation of *intrinsic* properties of syntactical objects. In contrast to this, our counter model shows that discontinuity is an *admissible* property which is not excluded by the syntax. In other words, continuity is not an essential property.

Structure of Part I

Part I is an extensive elaboration of Barendsen and Bezem (1992) (see (1991) for an extended abstract). Chapter 2 describes the extension of untyped λ-calculus with higher-type oracles, together with a proof of the Church-Rosser

property. Chapter 3 provides the necessary syntactical and semantical background on typed λ-calculi and their extensions with quantifier-free arithmetic and propositional logic. It contains a detailed description of the per construction and of the second-order extension of first-order models. Chapter 4 presents the results about bar recursion and polymorphism. The proof of the positive realizability result for $\boldsymbol{\lambda}$**TB** results moreover in direct non-conservativity proofs for extensions of $\boldsymbol{\lambda}$**T** with bar recursion, fan functional and Luckhardt's minimization functional, respectively.

Part II: Graph Rewrite Systems

The concept of graph rewriting is a relatively new model of computation that can serve as a theoretical basis for functional programming. More traditional examples of such models are the lambda calculus and term rewrite systems (TRS's, see Klop (1992)). A graph theoretical variant of the lambda calculus has been introduced by Wadsworth (1971). In this work we focus on a graphical variant of term rewriting.

Graph rewriting covers many aspects of functional programming, and fits more closely to the actual implementation of such languages. This is demonstrated by the 'intermediate language' *Clean* which is based on graph rewriting (see Brus et al. (1987) and Plasmeijer and van Eekelen (1993)). Theoretical results obtained in graph rewriting can often be implemented in a direct way. Moreover, graph rewrite systems proved to be a suitable abstraction to investigate communication between loosely coupled processors; see Barendsen and Smetsers (1992).

Graph rewrite systems (in the form that we consider) have been introduced in Barendregt et al. (1987b). We deal with a restricted form of these systems: the so-called *term graph rewrite systems* (TGRS's, see Barendregt et al. (1987a)). These graph rewrite systems have many aspects in common with TRS's: the specification (by rewrite rules) and application of functions (in objects) is separated, and the rewrite rules for functions may contain patterns. An additional feature of graph rewriting is *sharing* of arguments. In its ultimate form ('self-sharing') this results in cyclic structures. As a consequence, the collection of graph objects cannot be described or analysed using induction (like in the case of (λ-)terms). This obviously complicates reasoning about graphs and computations.

Barendsen and Smetsers (1992) (see also (1994)) developed basic tools for analysis of graph rewriting and applied these to investigate the combination of rewriting and de-sharing. For alternative foundations, see Ariola and Klop (1993) (equational rewriting) and Ehrig et al. (1987) (categorical formulations).

The work on *conventional typing* has been inspired by van Bakel et al. (1992) who consider TRS's. Our type system is essentially first-order: the types are built up from type variables, the standard type constructor $\rightarrow$, and other type constructors (like List) that can be introduced via *algebraic type specifications*.

An example of such a specification is

$$\mathrm{List}(\alpha) = \mathbf{Nil} \mid \mathbf{Cons}(\alpha, \mathrm{List}(\alpha)),$$

which specifies the type constructor List, and moreover the constructors **Nil** and **Cons** (for building list objects and defining functions by pattern matching).

The symbols of graph rewrite systems are supplied with a type by a *type environment*. This environment contains declarations $\mathbf{F} : (\tau_1, \ldots, \tau_k) \rightarrowtail \tau$, where k is the arity of **F**. The constructor part of this environment is determined by the algebraic specifications; for lists, e.g., one has

$$\begin{array}{rcl} \mathbf{Nil} & & \mathrm{List}(\alpha), \\ \mathbf{Cons} & : & (\alpha, \mathrm{List}(\alpha)) \rightarrowtail \mathrm{List}(\alpha). \end{array}$$

Due to the separation of specifications (rewrite rules, algebraic types) from applications one needs an *instantiation mechanism* to deal with different occurrences of symbols. For example, for the identity function symbol **I** (with obvious rewrite rule), a type environment would normally contain $\mathbf{I} : \alpha \rightarrowtail \alpha$. This type is considered as schematic: at an applicative occurrence of **I** one can conclude $\mathbf{I} : \tau \rightarrowtail \tau$ for each τ. This mechanism is sometimes called polymorphism, but has little to do with second-order type systems. In λ-calculus, one does not need an instantiation mechanism since different occurrences of the identity *term* $\lambda x.x$ can be typed separately.

The use of (higher-order) functionals in TGRS's (in which each symbol has a fixed arity) is made possible by adding a *Currying* procedure. Our results can be applied to ordinary rewrite systems and purely applicative ones (with application as only function), as well as to hybrid variants (such as used in *Clean*)

The notion of typing cannot be defined inductively, but is specified in terms of local requirements for a type assignment to nodes. In an application of a function **F**, say, these relate the type of the **F**-node to its environment type and the types of its arguments. The rewrite rules (which have a graph-like structure) are handled similarly.

For a fixed environment, each typable graph has a *principal type*, of which all other valid types are instances.

We show that typing is preserved during reduction (this is the so-called *subject reduction* property). It is possible to formulate a simplified typing concept for rewrite rules such that the subject reduction property still holds. The class of 'safe' rewrite systems is reasonably large and includes the graph-equivalents of familiar programming languages such as *Miranda*.

The research on *uniqueness typing* has been inspired by ideas originating from the research group on functional programming languages at the University of Nijmegen. The research reported in this thesis is the result of a translation of these ideas into a proper typing system.

Uniqueness typing is meant to solve two problems in implementations of functional programming languages: incorporation of non-functional operations and efficient space management. This will become clear below; we first go into the first topic since that provides the clearest intuition.

An important aspect of functional programming is *referential transparency:* each (sub)object represents a value, independent of its context. This is related to the Church-Rosser property: the order of evaluation does not influence the result of a computation. Therefore, multiple occurrences of the same (input) object refer to the same value. The addition of operations with inherent destructive effects on input objects (such as manipulations on a disk file) might violate this property

A first solution would be to allow only one single occurrence of such objects in a program. Via the propositions-as-types interpretation, this is related to *linear logic* (see Girard et al. (1989), Troelstra (1992)), since input parameters correspond to logical assumptions. The inputs are regarded as *resources*. Some ('linear') resources can be used exactly once; others ('unlimited resources') can be discarded, or used more than once. In linear logic, this is accomplished by adding a 'resource duplication' operator !. (The logic without ! will be referred to as *pure* linear logic.) A linear typing system for λ-calculus can be found in Wadler (1990) Our version (for arbitrary graphs) is somewhat more involved.

A minor modification concerns discarding of linear input, which one allows in uniqueness typing. Therefore, uniqueness typing corresponds closer to so-called *affine* logic, in which each assumption can be used *at most* once.

A first approach is to regard 'physically unique' objects, such as files, as linear resources (unique objects, in our terminology) and others as non-unique. Multiple occurrences correspond to multiple *references* to objects in graphs. As a consequence, the correctness of type assignment becomes dependent on the global reference structure of graphs, as opposed to the local function-argument dependency in the case of conventional typing. As in the conventional system, the object type of a (sub) graph is determined by the result type of its topmost symbol.

The idea is to attach *uniqueness attributes* to conventional types, with $\bullet$ standing for *unique*, and $\times$ for *non-unique*. The attribute $\times$ should not be confused with the linear duplication operator !, which produces (duplicatewise unique) copies. We do not have ! but consider $\times$-types for *non-linear* objects without any constraints on the number of references.

In the types of functions, it can now be specified that a given argument should be unique. In any application, the concrete function argument should have reference count 1, so the function has indeed 'private' access to its argument. This is used for destructive operations: e.g , **WriteChar** : $(\mathrm{Char}^{\times}, \mathrm{File}^{\bullet}) \rightarrowtail \mathrm{File}^{\bullet}$.

Sometimes, the uniqueness of unique objects is inessential, e.g., in the case of non-destructive functions like **ReadChar** : $\mathrm{File}^{\times} \rightarrowtail \mathrm{Char}^{\times}$, which can also be applied to a unique object (with reference count 1) Also multiple read access

to the same object file can be allowed The technical tool to achieve this is a subtyping relation (stating, e g , File$^{\bullet} \leq$ File$^{\times}$) in combination with the reference counting mentioned earlier, in the following way The reference count of an object and its type determine its 'access type' In the case of a single reference, the access type is the object type itself, like File$^{\bullet}$ in the above example In the case of multiple references, the access type becomes File$^{\times}$, indicating that the object cannot be used as unique The new typing requirement states that the access type of an actual function argument should be a *subtype* of the type requested by the function One easily checks that (again in our example) this allows both single and multiple read references, but only single write references (since File$^{\times} \not\leq$ File$^{\bullet}$) to a file

This mechanism is still too restrictive for practical use, however In many cases, depending on the evaluation strategy, one knows that a certain reference will be used (and disappear) before another reference to the same object In case these are the only references, one may regard the first one as 'multiple' and the second as 'single', and adjust the typing rules accordingly We present a refined dependency analysis of references in graphs, based on this observation

The relation with linear typing can thus be summarized as follows

$$\begin{aligned} \text{uniqueness typing} \;=\; & \text{pure linear typing} \\ & + \text{subtyping} \\ & + \text{strategy-aware reference analysis} \end{aligned}$$

Apart from the incorporation of destructive updates, uniqueness typing can also help to improve storage management If it is clear from a rewrite rule that a certain function argument is discarded, and it is specified as unique, then it will certainly become obsolete after rewriting Re-usage of its space can therefore be anticipated This leads to so-called *compile-time garbage collection*

In the context of graph rewriting with arbitrary sharing one encounters some delicate points that complicate the analysis of the typing system Firstly, in the presence of patterns even the simplest reference count analysis becomes subtle because functions do not only have access to their direct arguments, but also to arguments-of-arguments We present a treatment of algebraic types to deal with this properly As a result, there are several 'uniqueness variants' of algebraic types, such as 'spine unique' lists (of which only the **Cons**-nodes have reference count 1) Secondly, the proof of the subject reduction property is very involved Both the type assigment and the reference analysis of a graph and a rewrite rule have to be related to the typing of the rewrite result Due to the possible presence of cycles in graphs this analysis is complicated

Structure of Part II

Part II is a slightly modified version of Barendsen and Smetsers (1993a) (see (1993b) for an extended abstract) Chapter 5 gives a summary of the basic

concepts of graph rewriting, and presents the results on conventional typing Chapter 6 contains an exposition of uniqueness typing, and a proof of the subject reduction property The system differs slightly from the one described above there is an extra uniqueness attribute Δ for *essentially unique* objects without a non-unique variant Moreover, the relation between reference structure, object type and requested argument type is not expressed using access types, but by giving two versions of the subtyping relation $\leq$ This chapter contains directions for reading, the (very technical) reference analysis for the subject reduction proof is separated from the rest

Part III: Numerations

Numeration theory, invented by Eršov (1973), intends to capture the notion of computability on non numeric objects Standard examples are the class of partial recursive functions, recursively enumerable sets, and β-equivalence classes of λ-terms

The crucial concept is *encoding* although manipulation of the objects themselves is difficult, one often has a method of enumerating these Then an operation on the objects is considered computable (or *effective*) if it corresponds to a recursive function on their codes

Well-known numerations are the set of partial recursive functions with their indices, and the closed term model of untyped λ calculus with the λ-calculus enumerator **E** For the numeration of partial recursive functions, the notion of effective operation leads to a characterization of higher-order computability This corresponds to the definition of type level 2 of the model HEO, by the results of Myhill and Shepherdson (1955)

Chapter 7 presents an investigation of fixed point theorems in λ-calculus and recursion theory in the context of numerations It turns out that the often mentioned analogy between (proofs of) the λ-calculus fixed point theorem and the recursion theorem can be made more precise by distinguishing between code dependent and code free formulations We also present a survey of some classical results in recursion theory, including the effective version of the First Recursion Theorem using numerations

Part I

Lambda Calculus

Chapter 2

Lambda Calculus with Higher-Type Oracles

2.1. Syntax

In this section we will extend untyped λ-calculus with higher type objects (typically functionals) from a type structure. The untyped applicative structure thus obtained will be used to construct second-order models using partial equivalence relations (see Section 3.4).

2.1.1. DEFINITION. (i) The set of *first-order types* (notation $\mathbb{T}_1$) is given by the abstract syntax

$$\mathbb{T}_1 = \mathbb{C} \mid \mathbb{T}_1 \to \mathbb{T}_1.$$

Here $\mathbb{C}$ is a set of *type constants*. In this work we take $\mathbb{C} = \{\mathbf{0}\}$.

(ii) The *height* (or *type level*) of $\sigma \in \mathbb{T}_1$ (notation $\mathrm{h}(\sigma)$) is defined inductively by

$$\begin{aligned} \mathrm{h}(\mathbf{0}) &= 0, \\ \mathrm{h}(\sigma \to \tau) &= \max(\mathrm{h}(\sigma) + 1, \mathrm{h}(\tau)). \end{aligned}$$

NOTATION. (i) We let $\to$ associate to the right, so $\sigma \to \tau \to \rho$ stands for $\sigma \to (\tau \to \rho)$.

(ii) Note that each first order type is of the form

$$\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \mathbf{0}.$$

Such an expression will usually be abbreviated by $\vec{\sigma} \to \mathbf{0}$. Note that for each i

$$\mathrm{h}(\sigma_i) < \mathrm{h}(\vec{\sigma} \to \mathbf{0}) \quad (= \max_i \mathrm{h}(\sigma_i) + 1).$$

2.1.2. DEFINITION. (i) A first-order *type structure* is a structure

$$\mathfrak{M} = \langle (\mathfrak{M}_\sigma)_{\sigma \in \mathbb{T}_1},\ (\mathrm{App}_{\sigma,\tau})_{\sigma,\tau \in \mathbb{T}_1} \rangle$$

such that

(1) each $\mathfrak{M}_\sigma$ is a non-empty set;
(2) for each $\sigma, \tau \in \mathbb{T}_1$

$$\mathrm{App}_{\sigma,\tau} : \mathfrak{M}_{\sigma\to\tau} \times \mathfrak{M}_\sigma \to \mathfrak{M}_\tau$$

is an *application function* (we write $a \cdot b$ or simply ab for $\mathrm{App}_{\sigma,\tau}(a,b)$).

(ii) $\mathfrak{M}$ is an *ω-structure* if $\mathfrak{M}_0 = \mathbb{N}$.

(iii) a is an *$\mathfrak{M}$-element* if $a \in \bigcup_{\sigma\in\mathbb{T}_1} \mathfrak{M}_\sigma$. This is usually loosely denoted as $a \in \mathfrak{M}$.

(iv) $\mathfrak{M}$ is extensional if for each $a, a' \in \mathfrak{M}_{\sigma\to\tau}$

$$\forall b \in \mathfrak{M}_\sigma \ [ab = a'b] \ \Rightarrow \ a = a'.$$

We will freely make use of vector notation like $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$.

The most common example of an extensional type structure is the structure of functionals over a given set.

2.1.3. DEFINITION. Let X be a non-empty set. The *full type structure over X* (notation $\mathfrak{M}(X)$) is defined by setting

$$\mathfrak{M}(X) = \langle (X_\sigma)_{\sigma\in\mathbb{T}_1}, \ (\cdot_{\sigma,\tau})_{\sigma,\tau\in\mathbb{T}_1} \rangle$$

where

$$\begin{aligned} X_0 &= X, \\ X_{\sigma\to\tau} &= X_\tau^{X_\sigma} \end{aligned}$$

and

$$f \cdot a = f(a).$$

Of course $\mathfrak{M}(\mathbb{N})$ is an extensional ω-type structure.

The idea of adding higher-type oracles to untyped λ-calculus originates from Kleene (1962). He used this idea to prove the equivalence between recursiveness and lambda definability in higher types on $\mathfrak{M}(\mathbb{N})$.

In the sequel, let $\mathfrak{M} = \langle (\mathfrak{M}_\sigma)_{\sigma\in\mathbb{T}_1}, \ (\mathrm{App}_{\sigma,\tau})_{\sigma,\tau\in\mathbb{T}_1} \rangle$ be an extensional ω-type structure.

2.1.4. DEFINITION. (i) For each $\mathfrak{M}$-element of higher type $a \in \mathfrak{M}_\sigma$ (with $\sigma \neq \mathbf{0}$), let $\boldsymbol{a}$ be a constant associated with a. The collection of these so-called *oracles* is denoted by $\mathcal{O}_\mathfrak{M}$.

(ii) The set of *$\lambda\mathfrak{M}$-terms* (notation $\Lambda\mathfrak{M}$) is defined by the following abstract syntax.

$$\Lambda\mathfrak{M} = V \mid \mathcal{O}_\mathfrak{M} \mid (\Lambda\mathfrak{M}\,\Lambda\mathfrak{M}) \mid (\lambda V.\Lambda\mathfrak{M}).$$

We use the same notational conventions for $\lambda\mathfrak{M}$-terms as we do for λ-terms; see Barendregt (1984). The set of *closed* $\lambda\mathfrak{M}$-terms is denoted by $\Lambda^o\mathfrak{M}$.

The principal reduction relation is β-reduction A second notion of reduction will be added to the system We suppose the reader is familiar with the concept of reduction relations and induced conversion relations (denoted by $\to_\beta$, $\twoheadrightarrow_\beta$, and $=_\beta$ for β-reduction)

Now we consider the constants in $\Lambda\mathfrak{M}$ as higher-type oracles Some care is needed the result of applying a constant of type, say, $(\mathbf{0}\to\mathbf{0})\to\mathbf{0}\to\mathbf{0}$ to a type $\mathbf{0}\to\mathbf{0}$ object is *not* an object of the corresponding type $\mathbf{0}\to\mathbf{0}$ Instead, oracles of type $(\mathbf{0}\to\mathbf{0})\to\mathbf{0}\to\mathbf{0}$ only give a result if supplied with both a type $\mathbf{0}\to\mathbf{0}$ and a type $\mathbf{0}$ object In view of the type $\mathbf{0}$ objects being the only 'observables' in calculations, and the oracles giving results of computations in one single step, this is only natural

As the basic 'observable' objects are natural numbers, we can use some standard representations of natural numbers in untyped λ-calculus Adding other basic types for observables (e g , booleans) can also be modelled using the standard representations of the corresponding objects We will restrict ourselves to the natural numbers We use the Church numerals to represent these

2 1 5 DEFINITION (i) Let $F, M \in \Lambda\mathfrak{M}$ For every $n \in \mathbb{N}$, the n-th *iteration* of F on M (notation $F^n(M)$) is defined inductively by

$$\begin{aligned} F^0(M) &\equiv M, \\ F^{n+1}(M) &\equiv F(F^n(M)) \end{aligned}$$

(ii) For each $n \in \mathbb{N}$, the n-th *Church numeral* is defined by

$$\ulcorner n \urcorner \equiv \lambda f x\ f^n(x)$$

It is well known that the system $(\ulcorner n \urcorner)_{n\in\mathbb{N}}$ is adequate with respect to recursive functions each total recursive function is λ-definable with respect to the Church numerals Not every adequate numeral system is suitable for the applications in Chapter 3, e g , the recursor (needed for the interpretation of $\boldsymbol{\lambda}\mathbf{T}$) is not definable with respect to the (adequate) unsolvable numeral system from Barendsen (1991) One can work safely with the Church numerals, however

2 1 6 DEFINITION For each $a \in \mathfrak{M}$ the *standard representation* of a (notation $\underline{a}$) is defined by

$$\begin{aligned} a &\equiv \ulcorner a \urcorner && \text{if } a \in \mathfrak{M}_0, \\ &\equiv \boldsymbol{a} && \text{otherwise} \end{aligned}$$

2 1 7 DEFINITION Let $\to_R$ be a notion of reduction (on $\Lambda\mathfrak{M}$), $A \in \Lambda\mathfrak{M}$, and $a \in \mathfrak{M}$, say $a \in \mathfrak{M}_{\vec{\sigma}\to 0}$ Then A is said to R *represent* a (notation $A \rhd_R a$) if for all $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$ one has

$$A\underline{\vec{b}} =_R \underline{a\vec{b}} \quad (\equiv \ulcorner a\vec{b} \urcorner)$$

In particular, $A \rhd_R n$ iff $A =_R \ulcorner n \urcorner$ This notion is extended to sequences $(\vec{A} \rhd_R \vec{a})$ in the obvious way

The aim is to construct a reduction relation $\to_{\mathfrak{M}}$ such that for any $a, \vec{b}$ and $\vec{B}$ (of appropriate types)

$$\boldsymbol{a}\vec{B} \to_{\mathfrak{M}} \ulcorner a\vec{b} \urcorner \qquad \text{if} \qquad \vec{B} \rhd_{\beta\mathfrak{M}} \vec{b}. \tag{$*$}$$

Here $\beta\mathfrak{M}$-reduction is defined by $\to_{\beta\mathfrak{M}} = \to_\beta \cup \to_{\mathfrak{M}}$. Note that if '$\vec{B} \rhd_{\beta\mathfrak{M}} \vec{b}$' would be replaced by '$\vec{B} \rhd_\beta \vec{b}$', the reduction $\to_{\mathfrak{M}}$ could immediately be obtained by so-called δ-reduction making some external function internal (see Barendregt (1984), Section 15.3). This corresponds to the notion of *oracle* in recursion theory. In $(*)$, however, the behaviour of the oracle is specified in terms of itself, since $\boldsymbol{a}$ can occur in $\vec{B}$.

Because of this impredicativity, the question arises whether or not $\to_{\mathfrak{M}}$ can be well defined. This is indeed the case. The idea is to generate $\to_{\mathfrak{M}}$ in stages, according to a certain inductive operator.

2.1.8. DEFINITION. Let $R \subseteq \Lambda\mathfrak{M} \times \Lambda\mathfrak{M}$ be a relation. The *compatible closure* of R (notation $\bar{R}$) is defined inductively as follows.

$$\begin{array}{lcl} M\ R\ M' & \Rightarrow & M\ \bar{R}\ M'; \\ M\ \bar{R}\ M' & \Rightarrow & MN\ \bar{R}\ M'N, \\ & & NM\ \bar{R}\ NM', \\ & & \lambda x.M\ \bar{R}\ \lambda x.M'. \end{array}$$

2.1.9. EXAMPLE. $\to_\beta = \overline{\{((\lambda x.M)N, M[x := N]) \mid M, N \in \Lambda\mathfrak{M}, x \in V\}}$.

2.1.10. DEFINITION. The operator $\Gamma_{\mathfrak{M}} : \wp(\Lambda\mathfrak{M} \times \Lambda\mathfrak{M}) \to \wp(\Lambda\mathfrak{M} \times \Lambda\mathfrak{M})$ is defined by

$$\Gamma_{\mathfrak{M}}(R) = \{(\boldsymbol{a}\vec{B}, \ulcorner a\vec{b} \urcorner) \mid a \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}, \vec{B} \in \Lambda\mathfrak{M}, \vec{b} \in \mathfrak{M}_{\vec{\sigma}}, \vec{B} \rhd_{\beta R} \vec{b}\}.$$

2.1.11. LEMMA. $\Gamma_{\mathfrak{M}}$ *is monotone. That is,*

$$R \subseteq R' \ \Rightarrow\ \Gamma_{\mathfrak{M}}(R) \subseteq \Gamma_{\mathfrak{M}}(R').$$

PROOF. By monotonicity of $\bar{}$ and the fact that $\rhd_{\beta R}$ is monotone in R. □

Now the stages in the inductive definition are defined by setting for each ordinal number ζ (cf. Hinman (1978))

$$\Gamma^\zeta_{\mathfrak{M}} = \Gamma_{\mathfrak{M}}(\bigcup\{\Gamma^\vartheta_{\mathfrak{M}} \mid \vartheta < \zeta\}). \tag{$**$}$$

Using the denotation $\Gamma^{(\zeta)}_{\mathfrak{M}} = \bigcup\{\Gamma^\vartheta_{\mathfrak{M}} \mid \vartheta < \zeta\}$, the expression $(**)$ becomes

$$\Gamma^\zeta_{\mathfrak{M}} = \Gamma_{\mathfrak{M}}(\Gamma^{(\zeta)}_{\mathfrak{M}}).$$

From the theory of inductive definitions we know that for some minimal ζ_0, $\Gamma_{\mathfrak{M}}^{\zeta_0}$ is the least fixed point of $\Gamma_{\mathfrak{M}}$, so

$$\Gamma_{\mathfrak{M}}^{\zeta_0} = \Gamma_{\mathfrak{M}}^{(\zeta_0)}$$

We define $\to_{\mathfrak{M}} = \Gamma_{\mathfrak{M}}^{\zeta_0}$ It is clear that this $\to_{\mathfrak{M}}$ indeed satisfies

$$\boldsymbol{a}\vec{B} \to_{\mathfrak{M}} \ulcorner a\vec{b} \urcorner \qquad \text{if} \qquad \vec{B} \rhd_{\beta\mathfrak{M}} \vec{b}$$

As a consequence, the standard representations behave in the following way, as expected

2 1 12 LEMMA *Let* $a \in \mathfrak{M}$ *Then* $a \rhd_{\beta\mathfrak{M}} a$

PROOF Note that $a \rhd_{\beta\mathfrak{M}} a$ iff $a\,\vec{b} =_{\beta\mathfrak{M}} \underline{a\vec{b}}$ One verifies the property for each $a \in \mathfrak{M}_\sigma$ by induction on the height of σ □

Recently, Jager and Strahm (1994) describe a method for generating a reduction relation based on a first-order applicative theory, using similar transfinite induction techniques (but without the underlining technique used in the next section)

The original reduction system of Kleene (1962) used more restrictive contraction rules (requiring for a redex $\boldsymbol{a}M$ that M is closed and in normal form) The resulting reduction system, however, is not suitable for our applications in the next chapter A system with less restrictive contraction rules (only requiring closedness) is described in van Draanen (1989) Our reduction relation is the most general there are no syntactical restrictions on the form of an oracle redex As a consequence, the proof of the Church-Rosser is more complicated than earlier ones Our proof (see the next section) has been inspired by van Draanen (1989)

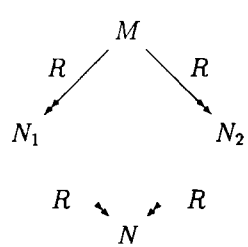
2.2. The Church-Rosser Property

In this section we will show that $\beta\mathfrak{M}$-reduction is Church-Rosser, or *confluent*

2 2 1 DEFINITION We say that a reduction relation $\to_R$ on $\Lambda\mathfrak{M}$ is *Church-Rosser* if for all $M, M_1, M_2 \in \Lambda\mathfrak{M}$

$$M \twoheadrightarrow_R M_1 \wedge M \twoheadrightarrow_R M_2 \;\Rightarrow\; \exists N\,[M_1 \twoheadrightarrow_R N \wedge M_2 \twoheadrightarrow_R N],$$

in a picture



The Church-Rosser property has two important consequences: normal forms are unique, and normal forms can be found by reduction: if M has R-normal form N, then $M \twoheadrightarrow_R N$.

2.2.2. MAIN THEOREM. *$\beta\mathfrak{M}$-reduction is Church-Rosser.*

The Main Theorem will be proved by showing for each ordinal ζ that the reduction relation $\to_\beta \cup \Gamma^\zeta_{\mathfrak{M}}$ is Church-Rosser, using transfinite induction.

To simplify notation, let $\to_\zeta$ abbreviate $\Gamma^\zeta_{\mathfrak{M}}$. Similarly one defines $\to_{\beta\zeta}$, $\to_{(\zeta)}$, $=_{\beta\zeta}$, etcetera.

The proof will occupy the rest of this section. We start with some auxiliary results.

2.2.3. SUBSTITUTION LEMMA. *Let $M, N, L \in \Lambda\mathfrak{M}$ and $x, y \in V$, such that $x \not\equiv y$ and $x \notin \mathrm{FV}(L)$. Then*

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

PROOF. By an easy induction on M. □

2.2.4. LEMMA. *Let $M, N \in \Lambda\mathfrak{M}$. If $M =_{\beta(\zeta)} N$, then for some $\vartheta < \zeta$ one has $M =_{\beta\vartheta} N$.*

PROOF. By induction on the generation of $=_{\beta(\zeta)}$. □

2.2.5. PROPOSITION. *The relation $=_{\beta\zeta}$ is substitutive, i.e.*

$$M =_{\beta\zeta} M' \;\Rightarrow\; M[x := N] =_{\beta\zeta} M'[x := N].$$

PROOF. By transfinite induction on ζ, and induction on the generation of $=_{\beta\zeta}$. We only treat the prime cases $\to_\beta$ and $\to_\zeta$. Let L^* denote $L[x := N]$. As to the (β) contraction rule, note that

$$\begin{array}{lcl} ((\lambda y.P)Q)^* & \equiv & (\lambda y.P^*)Q^* \\ & \to_\beta & P^*[y := Q^*] \\ & \equiv & (P[y := Q])^*, \quad \text{by the substitution lemma.} \end{array}$$

Now consider $\boldsymbol{a}\vec{B} \to_\zeta \ulcorner a\vec{b} \urcorner$, where $\vec{B} \rhd_{\beta(\zeta)} \vec{b}$. We claim that $\vec{B}^* \rhd_{\beta(\zeta)} \vec{b}$. As to component i, say $b_i \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}$. Let $\vec{c} \in \mathfrak{M}_{\vec{\sigma}}$. Then

$$B_i\underline{\vec{c}} =_{\beta(\zeta)} \ulcorner b_i\vec{c} \urcorner$$

so by Lemma 2.2.4, for some $\vartheta < \zeta$

$$B_i\underline{\vec{c}} =_{\beta\vartheta} \ulcorner b_i\vec{c} \urcorner.$$

Therefore by the induction hypothesis (of the transfinite induction)

$$(B_i\vec{c})^* =_{\beta\vartheta} (\ulcorner b_i\vec{c}\urcorner)^*,$$

which means

$$B_i^*\vec{c} =_{\beta\vartheta} \ulcorner b_i\vec{c}\urcorner.$$

This proves the claim. Now $\boldsymbol{a}\vec{B}^* \to_\zeta \ulcorner a\vec{b}\urcorner$ and we are done. □

2.2.6. COROLLARY. *$\rhd_{\beta\zeta}$ is substitutive, i.e.*

$$A \rhd_{\beta\zeta} a \;\Rightarrow\; A[x := N] \rhd_{\beta\zeta} a.$$

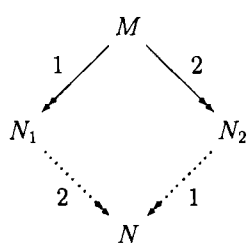
Now we can show that $\to_{\beta\zeta}$ is Church-Rosser for each ζ, by a transfinite induction.

In the sequel, let ζ be an ordinal number, and suppose $\beta\vartheta$-reduction is Church-Rosser for all $\vartheta < \zeta$. We will use 'main induction hypothesis' to refer to this assumption.

We use a method due to Hindley (1964) and Rosen (1973).

2.2.7. HINDLEY-ROSEN LEMMA. *Let $\to_1$ and $\to_2$ be reduction relations. Suppose*

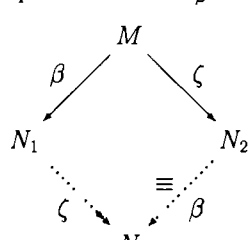
(1) *$\to_1$ is Church-Rosser,*
(2) *$\to_2$ is Church-Rosser,*
(3) *$\twoheadrightarrow_1$ and $\twoheadrightarrow_2$ commute, i.e.*



Then $\to_1 \cup \to_2$ is Church-Rosser.

PROOF. By an easy diagram chase. □

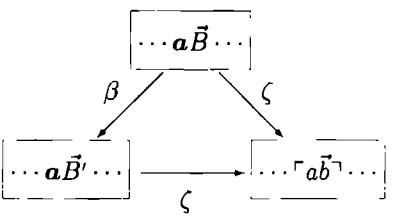
2.2.8. LEMMA. *Let $M \overset{=}{\to}_\beta N$ express that $M \to_\beta N$ or $M \equiv N$. Then*



PROOF. Say $M \overset{\Delta_1}{\to}_\beta N_1$, $M \overset{\Delta_2}{\to}_\zeta N_2$ where $\Delta_1 \equiv (\lambda x.P)Q$, $\Delta_2 \equiv \boldsymbol{a}\vec{B}$ with $\vec{B} \rhd_{\beta(\zeta)} \vec{b}$. (This is the 'most complex situation' for $\to_\zeta$; the case that $M \to N_2$ according to (ζ) is treated similarly.) We distinguish cases as to the relative positions of Δ_1 and Δ_2 in M.

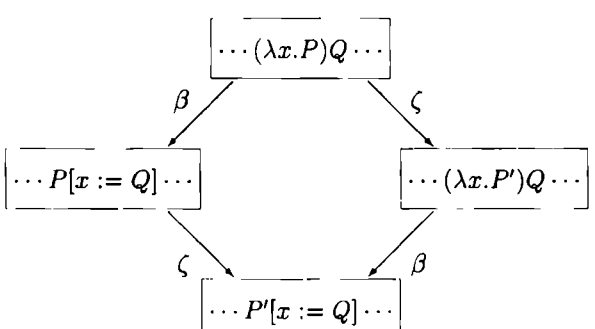
Case 1. Δ_1 and Δ_2 are disjoint. Then the result is trivial.

Case 2. $\Delta_1 \subseteq \Delta_2$. Let $\vec{B}'$ be the result of contacting Δ_2 in $\vec{B}$. Then $\vec{B}' \rhd_{\beta(\zeta)} \vec{b}$ since $\vec{B} =_\beta \vec{B}'$ so the situation is as follows.



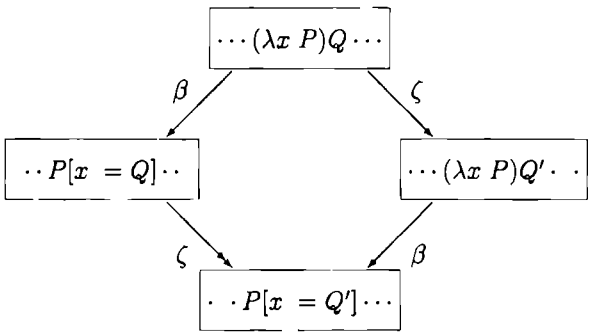
Case 3. $\Delta_2 \subseteq \Delta_1$. Then either $\Delta_2 \subseteq P$ or $\Delta_2 \subseteq Q$.

Case 3a. $\Delta_2 \subseteq P$. Let P' be the result of contracting Δ_2 in P. Then



is a correct diagram since $\vec{B}[x := Q] \rhd_{\beta(\zeta)} \vec{b}$ by Corollary 2.2.6.

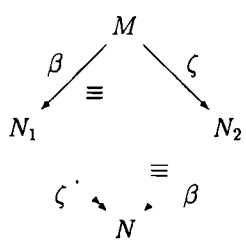
Case 3b $\Delta_2 \subseteq Q$ Let Q' be the result of contracting Δ_2 in Q Then one has



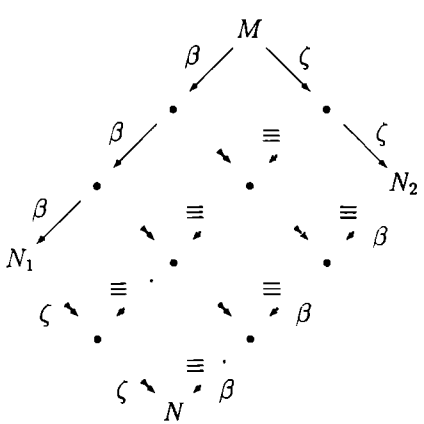
where the number of steps in the $\twoheadrightarrow_\zeta$ reduction depends on the multiplicity of x in P □

2 2 9 PROPOSITION $\twoheadrightarrow_\beta$ *and* $\twoheadrightarrow_\zeta$ *commute*

PROOF. By Lemma 2 2 8 one has



Now by a diagram chase suggested in the following figure we are done



□

In the following we will show that $\to_\zeta$ is Church-Rosser. We first prove some facts about representation of elements in the type structure.

2.2.10. NON-AMBIGUITY LEMMA. (i) *Let* $b_1, b_2 \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}$. *Then*

$$B \rhd_{\beta(\zeta)} b_1,\ B \rhd_{\beta(\zeta)} b_2 \ \Rightarrow\ b_1 = b_2.$$

(ii) *Let* $n_1, n_2 \in \mathbb{N}$. *Then*

$$\boldsymbol{a}\vec{B} \to_\zeta \ulcorner n_1 \urcorner,\ \boldsymbol{a}\vec{B} \to_\zeta \ulcorner n_2 \urcorner \ \Rightarrow\ n_1 = n_2.$$

PROOF. (i) Suppose $B \rhd_{\beta(\zeta)} b_1$ and $B \rhd_{\beta(\zeta)} b_2$. Let $\vec{c} \in \mathfrak{M}_{\vec{\sigma}}$. Then

$$B\underline{\vec{c}} =_{\beta(\zeta)} \ulcorner b_1\vec{c} \urcorner,$$

so for some $\vartheta_1 < \zeta$ one has

$$B\underline{\vec{c}} =_{\beta\vartheta_1} \ulcorner b_1\vec{c} \urcorner.$$

Similarly one shows that for some $\vartheta_2 < \zeta$

$$B\underline{\vec{c}} =_{\beta\vartheta_2} \ulcorner b_2\vec{c} \urcorner.$$

Now suppose (without loss of generality) $\vartheta_1 \le \vartheta_2$. Then also $B\vec{c} =_{\beta\vartheta_2} \ulcorner b_1\vec{c} \urcorner$. Note that numerals are normal forms; hence $\ulcorner b_1\vec{c} \urcorner \equiv \ulcorner b_2\vec{c} \urcorner$ by the Church-Rosser property for $\beta\vartheta_2$-reduction. Now by extensionality of $\mathfrak{M}$ the result follows.

(ii) By (i). □

2.2.11. COROLLARY. $\rhd_{\beta(\zeta)}$ *can be considered as a partial mapping.*

In order to keep track of specific ζ-redexes during reduction, we introduce a kind of *underlining* similar to the indexing employed by Barendregt (1984) in the proof of the Church-Rosser property for β-reduction.

2.2.12. DEFINITION (Marking). (i) The set of terms with *marked* ζ*-redexes* (notation $\underline{\Lambda}\underline{\mathfrak{M}}$) is defined inductively as follows. Below, M^- denotes M after removal of all markings; this is defined simultaneously.

$$\begin{aligned}
x \in V &\ \Rightarrow\ x \in \underline{\Lambda}\underline{\mathfrak{M}},\\
a \in \mathfrak{M}\backslash\mathfrak{M}_{\mathbf{0}} &\ \Rightarrow\ \boldsymbol{a} \in \underline{\Lambda}\underline{\mathfrak{M}},\\
M, N \in \underline{\Lambda}\underline{\mathfrak{M}} &\ \Rightarrow\ (MN) \in \underline{\Lambda}\underline{\mathfrak{M}},\\
M \in \underline{\Lambda}\underline{\mathfrak{M}}, x \in V &\ \Rightarrow\ (\lambda x.M) \in \underline{\Lambda}\underline{\mathfrak{M}},\\
\left.\begin{array}{r} a \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}, \vec{B} \in \underline{\Lambda}\underline{\mathfrak{M}} \\ \exists \vec{b} \in \mathfrak{M}_{\vec{\sigma}}\ [\vec{B}^- \rhd_{\beta(\zeta)} \vec{b}] \end{array}\right\} &\ \Rightarrow\ (\boldsymbol{a}\vec{B}) \in \underline{\Lambda}\underline{\mathfrak{M}}.
\end{aligned}$$

Moreover

$$\begin{aligned} x^- &\equiv x, \\ \boldsymbol{a}^- &\equiv \boldsymbol{a}, \\ (MN)^- &\equiv M^-N^-, \\ (\lambda x.M)^- &\equiv \lambda x.M^-, \\ (\underline{\boldsymbol{a}}\vec{B})^- &\equiv \boldsymbol{a}\vec{B}^-. \end{aligned}$$

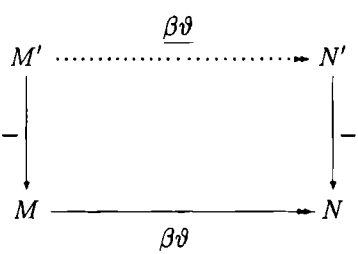
(ii) For each $\vartheta \leq \zeta$, the notions of reduction $\to_{\underline{\vartheta}}$, $\to_{\underline{\beta\vartheta}}$ are defined in the same way as before, giving $\underline{\boldsymbol{a}}$ exactly the same behaviour as $\boldsymbol{a}$. So the contraction rules are

$$\begin{aligned} \boldsymbol{a}\vec{B} &\to \ulcorner a\vec{b}\urcorner \quad \text{if } \vec{B}^- \rhd_{\beta(\vartheta)} \vec{b}, \\ \underline{\boldsymbol{a}}\vec{B} &\to \ulcorner a\vec{b}\urcorner \quad \text{if } \vec{B}^- \rhd_{\beta(\vartheta)} \vec{b}. \end{aligned}$$

Notice that $\Lambda\underline{\mathfrak{M}}$ is closed under reduction by Corollary 2.2.6.

Observing that $\underline{\boldsymbol{a}}\vec{B}$ and $\boldsymbol{a}\vec{B}$ act the same, it is not hard to believe that $\to_{\underline{\beta\vartheta}}$ is Church-Rosser for each $\vartheta < \zeta$. This is made precise in the following technical results. Below, $M \bar{\to} N$ expresses that $M^- \equiv N$.

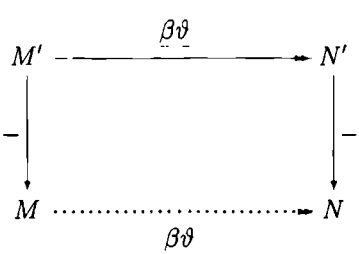
2.2.13. LEMMA (Lifting).



$M, N \in \Lambda\underline{\mathfrak{M}}$,
$M', N' \in \underline{\Lambda\underline{\mathfrak{M}}}$.

PROOF. First consider a one step reduction. Then N' is obtained by contracting the corresponding redex in M'. The general statement follows by transitivity. □

2.2.14. LEMMA (Projecting).

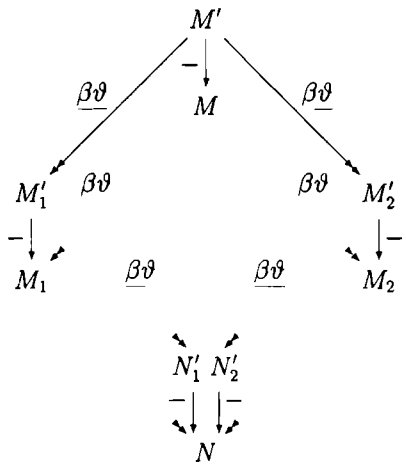


$M', N' \in \underline{\Lambda\underline{\mathfrak{M}}}$,
$M, N \in \Lambda\underline{\mathfrak{M}}$.

PROOF Obvious □

2 2 15 PROPOSITION *$\underline{\beta\vartheta}$ reduction is Church-Rosser for each $\vartheta < \zeta$*

PROOF By the main induction hypothesis, $\beta\vartheta$-reduction is Church-Rosser Now by lifting and projecting reduction sequences we can erect the following diagram



With a little thought one sees that N_1' and N_2' must be syntactically equal (oth erwise trace back differently marked redexes) □

We use denotations $=_{\underline{\beta\vartheta}}$ and $=_{\underline{\beta(\vartheta)}}$ like for the unmarked system Results for $\to_{\beta\vartheta}$ can easily be translated into corresponding ones for the marked systems

2 2 16 DEFINITION Let $M \in \underline{\Lambda\mathfrak{M}}$ Then $\Phi(M) \in \Lambda\mathfrak{M}$ is defined as follows

$$\begin{aligned}
\Phi(x) &\equiv x,\\
\Phi(\boldsymbol{a}) &\equiv \boldsymbol{a},\\
\Phi(MN) &\equiv \Phi(M)\Phi(N),\\
\Phi(\lambda x\, M) &\equiv \lambda x\, \Phi(M),\\
\Phi(\underline{\boldsymbol{a}}\vec{B}) &\equiv \ulcorner a\vec{b}\urcorner, \quad \text{if } \vec{B}^- \rhd_{\beta(\zeta)} \vec{b}
\end{aligned}$$

So Φ contracts marked ζ-redexes Note that Φ is well-defined (use the Non-ambiguity Lemma in the last clause)

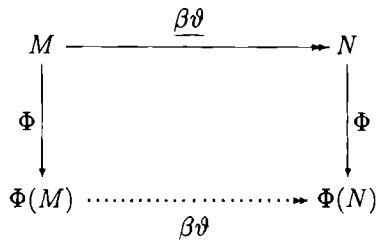
2 2 17 LEMMA $\Phi(M[x := N]) \equiv \Phi(M)[x := \Phi(N)]$

PROOF Induction on M The cases x, $\boldsymbol{a}$, M_1M_2, and $\lambda x\, M_1$ are easy Now consider $M \equiv \underline{\boldsymbol{a}}\vec{B}$, with $\vec{B}^- \rhd_{\beta(\zeta)} \vec{b}$ By Corollary 2 2 6 also $\vec{B}^-[x := N] \rhd_{\beta(\zeta)} \vec{b}$, so

$$\begin{aligned}
\Phi(\underline{\boldsymbol{a}}\vec{B}[x := N]) &\equiv \ulcorner a\vec{b}\urcorner\\
&\equiv \Phi(\underline{\boldsymbol{a}}\vec{B})[x := \Phi(N)] \quad \square
\end{aligned}$$

In diagrams we use $M \overset{\Phi}{\twoheadrightarrow} N$ to indicate that $\Phi(M) \equiv N$.

2.2.18. LEMMA. *For each* $\vartheta < \zeta$



$M, N \in \underline{\Lambda\mathfrak{M}}$.

PROOF. By transfinite induction on ϑ. Suppose $\vartheta < \zeta$ and the statement holds for all $\vartheta' < \vartheta$. We proceed by induction on the generation of $\twoheadrightarrow_{\underline{\beta\vartheta}}$. First consider a one step reduction $\to_{\underline{\beta\vartheta}}$.

Case 1. $M \to_{\underline{\beta\vartheta}} N$ is $(\lambda x.P)Q \to_\beta P[x := Q]$. Then we are done by Lemma 2.2.17.

Case 2. $M \to_{\underline{\beta\vartheta}} N$ is $\boldsymbol{a}\vec{B} \to \ulcorner a\vec{b}\urcorner$ where $\vec{B}^- \rhd_{\beta(\vartheta)} \vec{b}$. Note that $\Phi(\boldsymbol{a}\vec{B}) \equiv \boldsymbol{a}\Phi(\vec{B})$.

Claim. $\Phi(\vec{B}) \rhd_{\beta(\vartheta)} \vec{b}$. Then we are done.

Proof. For all i and $\vec{c}$ (of appropriate types) one has by assumption

$$B_i^- \underline{\vec{c}} =_{\beta(\vartheta)} \ulcorner b_i \vec{c}\urcorner,$$

so for some $\vartheta' < \vartheta$

$$B_i^- \underline{\vec{c}} =_{\beta\vartheta'} \ulcorner b_i \vec{c}\urcorner.$$

Therefore $B_i^- \underline{\vec{c}} \twoheadrightarrow_{\beta\vartheta'} \ulcorner b_i \vec{c}\urcorner$ by the main induction hypothesis, so $B_i \underline{\vec{c}} \twoheadrightarrow_{\underline{\beta\vartheta'}} \ulcorner b_i \vec{c}\urcorner$ by lifting. Now by the hypothesis of the transfinite induction

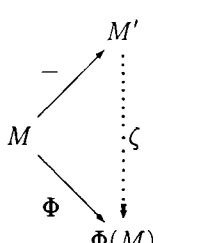
$$\Phi(B_i \underline{\vec{c}}) \twoheadrightarrow_{\beta\vartheta'} \Phi(\ulcorner b_i \vec{c}\urcorner) \equiv \ulcorner b_i \vec{c}\urcorner.$$

Hence also $\Phi(B_i)\underline{\vec{c}} =_{\beta(\vartheta)} \ulcorner b_i \vec{c}\urcorner$. $\square_{Claim}$

Case 3. $M \to_{\underline{\beta\vartheta}} N$ is $\underline{\boldsymbol{a}}\vec{B} \to \ulcorner a\vec{b}\urcorner$, where $\vec{B}^- \rhd_{\underline{\beta(\vartheta)}} \vec{b}$. Then $\Phi(\underline{\boldsymbol{a}}\vec{B}) \equiv \ulcorner a\vec{b}\urcorner$ and we are done.

The other cases (regarding the compatibility rules) are easy. Moreover, the general statement follows by transitivity. $\square$

2.2.19. LEMMA.

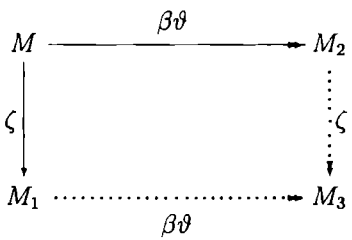


$M \in \underline{\Lambda\mathfrak{M}}$,
$M' \in \Lambda\mathfrak{M}$.

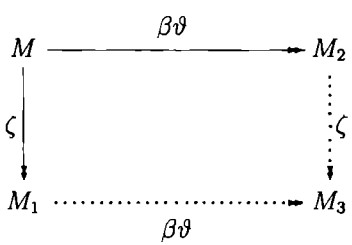
PROOF. Induction on the structure of M. The most interesting case is $M \equiv \underline{a}\vec{B}$ where $\vec{B}^- \rhd_{\beta(\zeta)} \vec{b}$. Then $\Phi(M) \equiv \ulcorner a\vec{b} \urcorner$ and $M^- \equiv \boldsymbol{a}\vec{B}^- \to_\zeta \ulcorner a\vec{b} \urcorner$, so we are done. □

2.2.20. PROPOSITION. *Let* $\vartheta < \zeta$.

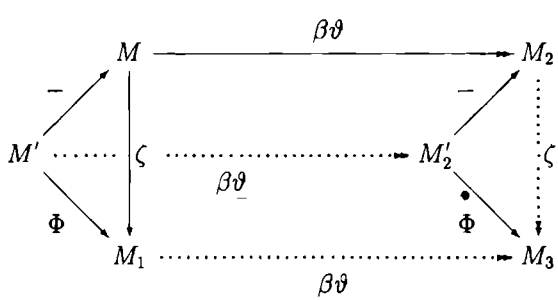
(i)



(ii)

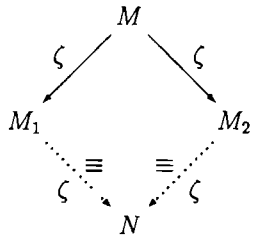


PROOF. (i) Say $M \xrightarrow{\Delta}_\zeta M_1$. Let $M' \in \underline{\Lambda\mathfrak{M}}$ be the term obtained by marking Δ in M. By the lemmas 2.2.13, 2.2.18 and 2.2.19 one can erect the following diagram.



(ii) By (i) and an easy diagram chase. □

2.2.21. LEMMA.



PROOF. Set $M \overset{\Delta_1}{\twoheadrightarrow} M_1$, $M \overset{\Delta_2}{\twoheadrightarrow} M_2$. Distinguish cases as to the relative positions of Δ_1 and Δ_2.

Case 1. Δ_1 and Δ_2 are disjoint. This case is trivial.

Case 2. Δ_1 and Δ_2 coincide. Then we are done by the Non-ambiguity Lemma 2.2.10.

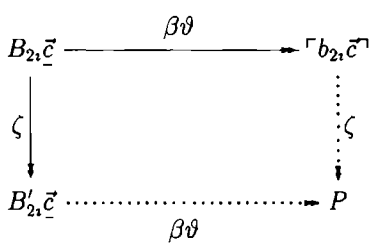
Case 3. $\Delta_1 \subseteq \Delta_2$, say $\Delta_1 \equiv \boldsymbol{a}_1\vec{B}_1$, $\vec{B}_1 \rhd_{\beta(\zeta)} \vec{b}_1$, $\Delta_2 \equiv \boldsymbol{a}_2\vec{B}_2$, $\vec{B}_2 \rhd_{\beta(\zeta)} \vec{b}_2$. Let $\vec{B}_2'$ be the result of replacing Δ_1 in $\vec{B}_2$ by $\ulcorner a_1\vec{b}_1\urcorner$. (In fact, this substitution affects only one B_{2i}.)

Claim. $\vec{B}_2' \rhd_{\beta(\zeta)} \vec{b}_2$.

Proof. As to B_{2i}', let $\vec{c}$ be of the appropriate types. Then

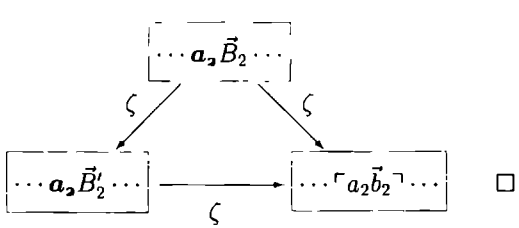
$$B_{2i}\underline{\vec{c}} =_{\beta\vartheta} \ulcorner b_{2i}\vec{c}\urcorner$$

for some $\vartheta < \zeta$, so by the main induction hypothesis (see Page 19) $B_{2i}\underline{\vec{c}} \twoheadrightarrow_{\beta\vartheta} \ulcorner b_{2i}\vec{c}\urcorner$. By Proposition 2.2.20 (ii) one has for some P



But $\ulcorner b_{2i}\vec{c}\urcorner$ is in ζ-nf so $P \equiv \ulcorner b_{2i}\vec{c}\urcorner$. Therefore $B_{2i}'\underline{\vec{c}} =_{\beta\vartheta} \ulcorner b_{2i}\vec{c}\urcorner$ and hence $B_{2i}'\underline{\vec{c}} =_{\beta(\zeta)} \ulcorner b_{2i}\vec{c}\urcorner$. $\square_{Claim}$

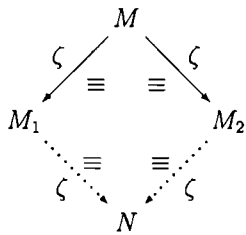
So we have



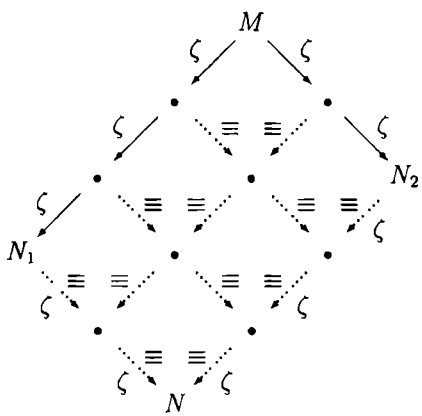
$\square$

2.2.22. PROPOSITION. *ζ-reduction is Church-Rosser.*

PROOF. By Lemma 2.2.21 one has



Now the following diagram chase shows that $\rightarrow_\zeta$ is Church-Rosser.



□

2.2.23. COROLLARY. *$\beta\zeta$-reduction is Church-Rosser.*

PROOF. By the Hindley-Rosen Lemma, using the propositions 2.2.9 and 2.2.22, and the fact that β-reduction is Church-Rosser. □

Now we have completed the main transfinite induction.

2.2.24. THEOREM. *$\beta\mathfrak{M}$-reduction is Church-Rosser.*

Chapter 3

First- and Second-Order Lambda Calculus

3.1. Syntax

In this section we describe the syntax of some systems of first-order lambda calculus and of their second-order extensions

The first-order system $\boldsymbol{\lambda}^{\tau}$, the so-called *simply typed lambda calculus* was introduced by Church (1940) Its second-order extension $\boldsymbol{\lambda 2}$ of *polymorphic lambda calculus* is due to Girard (1972)

The first and second order systems are distinguished by their respective sets of types, $\mathbb{T}_1$, $\mathbb{T}_2$ The set $\mathbb{T}_1$ has been introduced before (see Section 2 1)

3 1 1 DEFINITION The sets of first and second order *types* are given by the following abstract syntax

$$\begin{aligned} \mathbb{T}_1 &= \mathbb{C} \mid \mathbb{T}_1 \to \mathbb{T}_1, \\ \mathbb{T}_2 &= \mathbb{C} \mid \mathbb{V} \mid \mathbb{T}_2 \to \mathbb{T}_2 \mid \forall \mathbb{V}\, \mathbb{T}_2 \end{aligned}$$

Here $\mathbb{V} = \{\alpha, \beta, \alpha', \quad\}$ is an infinite set of *type variables*, and $\mathbb{C}$ is a set of *type constants* In this work we take $\mathbb{C} = \{\mathbf{0}\}$ Types from $\mathbb{T}_1$ are sometimes called *simple* types and types from $\mathbb{T}_2$ *polymorphic* types Note that $\mathbb{T}_1 \subset \mathbb{T}_2$ In the sequel, $\sigma, \tau, \sigma', \quad$ range over types. It is convenient to single out some special simple types called *pure types* $\mathbf{1} \equiv \mathbf{0} \to \mathbf{0}$, $\mathbf{2} \equiv \mathbf{1} \to \mathbf{0}$, and so on

The terms of the systems are built from typed variables and typed constants (specific for each system), using application and lambda abstraction Below, $\boldsymbol{\lambda}\Box$ ranges over systems

3 1 2 DEFINITION (i) For each type σ, let

$$\mathrm{Var}_\sigma = \{v_0^\sigma, v_1^\sigma, \quad\}$$

be an infinite set of *variables of type* σ Below, $x, y, z, \quad$ range over variables, their respective types are indicated by superscripts (x^σ) when necessary

(ii) For each type σ we assume a set $\mathrm{Cons}_\sigma(\boldsymbol{\lambda}\Box)$ of *constants of type* σ to be given. For the base systems $\boldsymbol{\lambda}^\tau$ and $\boldsymbol{\lambda 2}$ one takes $\mathrm{Cons}_\sigma(\boldsymbol{\lambda}\Box) = \emptyset$ for each σ.

(iii) For each $\sigma \in \mathbb{T}_1, \mathbb{T}_2$ respectively, the set of $\boldsymbol{\lambda}\Box$-*terms of type* σ (notation $\mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box)$) is defined inductively as follows.

$$\begin{aligned} x^\sigma \in \mathrm{Var}_\sigma &\Rightarrow x^\sigma \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box), \\ c \in \mathrm{Cons}_\sigma(\boldsymbol{\lambda}\Box) &\Rightarrow c \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box), \\ M \in \mathrm{Term}_{\sigma\to\tau}(\boldsymbol{\lambda}\Box),\ N \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box) &\Rightarrow (MN) \in \mathrm{Term}_\tau(\boldsymbol{\lambda}\Box), \\ M \in \mathrm{Term}_\tau(\boldsymbol{\lambda}\Box),\ x^\sigma \in \mathrm{Var}_\sigma &\Rightarrow (\lambda x^\sigma.M) \in \mathrm{Term}_{\sigma\to\tau}(\boldsymbol{\lambda}\Box). \end{aligned}$$

For the second order systems we also include

$$\begin{aligned} M \in \mathrm{Term}_{\forall\alpha\,\sigma}(\boldsymbol{\lambda}\Box),\ \tau \in \mathbb{T}_2 &\Rightarrow (M\tau) \in \mathrm{Term}_{\sigma[\alpha:=\tau]}(\boldsymbol{\lambda}\Box), \\ M \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box),\ \alpha \in \mathbb{V} &\Rightarrow (\Lambda\alpha.M) \in \mathrm{Term}_{\forall\alpha\,\sigma}(\boldsymbol{\lambda}\Box). \end{aligned}$$

(iv) The set of $\boldsymbol{\lambda}\Box$-*terms* is

$$\mathrm{Term}(\boldsymbol{\lambda}\Box) = \bigcup_\sigma \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box).$$

For $\boldsymbol{\lambda}\Box$-terms we use the denotational conventions of Barendregt (1984).

3.1.3. EXAMPLE. (i) Typical examples of $\boldsymbol{\lambda}^\tau$-terms are

$$\begin{aligned} x^{\mathbf{0}} &\text{ of type } \mathbf{0}, \\ f^{\mathbf{1}} &\text{ of type } \mathbf{1}, \\ \lambda f^{\mathbf{1}} x^{\mathbf{0}}.f(fx) &\text{ of type } \mathbf{1}{\to}\mathbf{0}{\to}\mathbf{0}, \\ (\lambda f^{\mathbf{1}} x^{\mathbf{0}}.f(fx))(\lambda y^{\mathbf{0}}.y) &\text{ of type } \mathbf{0}{\to}\mathbf{0} \quad (\equiv \mathbf{1}). \end{aligned}$$

(ii) Typical examples of $\boldsymbol{\lambda 2}$-terms are

$$\begin{aligned} \Lambda\alpha.\lambda x^\alpha.x &\text{ of type } \forall\alpha.\alpha{\to}\alpha, \\ \Lambda\alpha.\lambda f^{\alpha\to\alpha} x^\alpha.f(fx) &\text{ of type } \forall\alpha.(\alpha{\to}\alpha){\to}\alpha{\to}\alpha, \\ (\Lambda\alpha.\lambda x^\alpha.x)\mathbf{1} &\text{ of type } \mathbf{1}{\to}\mathbf{1}. \end{aligned}$$

Here and below we assume familiarity with notions such as $\mathrm{FV}(M)$, the set of free variables of M, *hygienic* substitution (expressed by $[x := N]$, $[\alpha := \tau]$ etc.), closed term, and so on. The usual care in dealing with variables should be exercised. For example, in $\Lambda\alpha.M$ we assume that the type variable α does not occur free in any type of a free term variable occurring in M. Furthermore we shall tacitly assume that terms are well-typed and shall reduce type superscripts to a minimum that is needed to reconstruct the types of the constituents of a well-typed term. Below, $F, \ldots, M, M', \ldots, Y$ range over terms.

For the moment, we focus on *equational* $\lambda\Box$ theories We view these systems in the first place as models of (higher-order, subrecursive) computation In this view the choice for equational theories is natural Moreover, equational theories exhibit an attractive conceptual simplicity In Chapter 4, we will consider extensions with quantifier-free arithmetic and with propositional logic

3 1 4 DEFINITION The *formulas* of $\lambda\Box$ are *equations* $M =_\sigma N$ with σ a type of $\lambda\Box$ and M and N terms of $\lambda\Box$ of type σ Typical examples of such equations can be found in the inference rules that define the theories $\lambda\Box$ below In the sequel, $E, E', E(x, y),$ range over equations, and (to smoothen future extensions) $\varphi, \varphi',$ over arbitrary formulas We shall omit the type subscript in equations when confusion is not likely

3 1 5 DEFINITION The *theories* $\lambda\Box$ are built up from axioms and rules that naturally divide into two groups

1 *Lambda calculus* axioms and rules We distinguish the following subgroups

(L1) Basic axioms and rules for simply typed lambda calculus (first order)
Primary axioms

$$(\lambda x^\sigma\, M)N =_\tau M[x := N] \qquad (\beta_1)$$

and

$$\lambda x^\sigma\, Mx =_{\sigma\to\tau} M, \qquad (\eta_1)$$

provided $x \notin \mathrm{FV}(M)$
Equality rules

$$M =_\sigma M \qquad \frac{M =_\sigma N}{N =_\sigma M} \qquad \frac{M =_\sigma N \qquad N =_\sigma L}{M =_\sigma L}$$

Compatibility rules

$$\frac{M =_{\sigma\to\tau} N}{ML =_\tau NL} \qquad \frac{M =_\sigma N}{FM =_\tau FN} \qquad \frac{M =_\tau N}{\lambda x^\sigma\, M =_{\sigma\to\tau} \lambda x^\sigma\, N} \ (\xi_1)$$

(L2) Axioms and rules for polymorphism (second order)
Primary axioms

$$(\Lambda\alpha\, M)\tau =_{\sigma[\alpha:=\tau]} M[\alpha := \tau] \qquad (\beta_2)$$

and

$$\Lambda\alpha\, M\alpha =_{\forall\alpha\,\sigma} M, \qquad (\eta_2)$$

provided that α is *loose* in M, i e not free in any type occuring in M (optional rule, not used in this work)

Equality and compatibility axioms/rules extended to the second order case, including:

$$\frac{M =_{\forall\alpha.\sigma} N}{M\tau =_{\sigma[\alpha:=\tau]} N\tau} \qquad \frac{M =_{\sigma} N}{\Lambda\alpha.M =_{\sigma\to\tau} \Lambda\alpha.N}\ (\xi_2)$$

2. Defining equations for *constants*, to be decribed separately for each system.

The corresponding *deduction relations* $\vdash_{\lambda\square}$ are defined as usual in natural deduction systems.

For some extensions of the systems described in this chapter, it is necessary to allow derivations to depend on *assumptions* ($\Gamma \vdash \varphi$ as opposed to just $\vdash \varphi$), which is no problem in a natural deduction system. This is not standard in lambda calculus; some additional care in dealing with variables should be exercised. The ξ-rule has to be restricted: $\lambda x.M = \lambda x.N$ can only be inferred from $M = N$ when x does not occur free in any assumption on which $M = N$ depends.

3.1.6. REMARK. The η-axiom is in fact equivalent to the following *rule of extensionality*.

$$\frac{Mx^{\sigma} =_{\tau} Nx^{\sigma}}{M =_{\sigma\to\tau} N}\ (\text{EXT})$$

Here x must neither occur in $\text{FV}(M)$, nor in $\text{FV}(N)$, nor in any assumption on which the premiss depends. The η-axiom will mostly be used in the form of the rule (EXT).

One of the best known extensions of $\boldsymbol{\lambda}^{\tau}$ is *Gödel's* $\mathbf{T}$, which results by adding constants for natural numbers and primitive recursion. We refer to this system as $\boldsymbol{\lambda}\mathbf{T}$. Its second order version $\boldsymbol{\lambda 2}\mathbf{T}$ is in fact equivalent to Girard's *Système* $\mathbf{F}$.

3.1.7. DEFINITION. (i) The systems $\boldsymbol{\lambda}\mathbf{T}$ (first order) and $\boldsymbol{\lambda 2}\mathbf{T}$ (second order) are based on $\boldsymbol{\lambda}^{\tau}$ (resp. $\boldsymbol{\lambda 2}$), but allow the typed *constants*

$$\boldsymbol{0}, \boldsymbol{S}, \boldsymbol{R}_{\sigma},$$

where $\boldsymbol{0}$ is of type $\mathbf{0}$ and $\boldsymbol{S}$ is of type $\mathbf{1}$, with intended interpretation *zero* respectively the *successor function* (the intended interpretation of $\mathbf{0}$ is the set of natural numbers). We use the abbreviations $\overline{0} \equiv \boldsymbol{0}$, $\overline{1} \equiv \boldsymbol{S0}$, $\overline{2} \equiv \boldsymbol{S}(\boldsymbol{S0})$, and so on. The constants $\boldsymbol{R}_{\sigma}$ of type $\sigma\to(\mathbf{0}\to\sigma\to\sigma)\to\mathbf{0}\to\sigma$ are added for all types σ of the system in question; their intended interpretation is that of *primitive recursor*.

(ii) The *rules* for these constants are the following.

$$\boldsymbol{R}_{\sigma}MN\boldsymbol{0} =_{\sigma} M \qquad \boldsymbol{R}_{\sigma}MN(\boldsymbol{S}P) =_{\sigma} NP(\boldsymbol{R}_{\sigma}MNP) \qquad (\text{CR})$$

A typical example of a $\lambda\mathbf{T}$-term is $\mathbf{Add} \equiv \lambda x^0 y^0.\boldsymbol{R}_0 x(\lambda z^0 p^0.\boldsymbol{S}p)y$. Then, e.g., $\mathbf{Add}\,\bar{2}\,\bar{3} =_{\mathbf{0}} \bar{5}$ is provable in $\lambda\mathbf{T}$.

Let us expand a bit on the relation between first-order and second-order systems.

3.1.8. DEFINITION. Let $\lambda\Box$ be a first-order system. The *plain polymorphic extension* of $\lambda\Box$ (notation $(\lambda\Box)^2$) is the second order system that has the same constants as $\lambda\Box$, and the rules of $\lambda\Box$ extended with (L2).

Obviously, $\boldsymbol{\lambda 2} = (\lambda^\tau)^2$. Note that, however, $\boldsymbol{\lambda 2}\mathbf{T} \neq (\lambda\mathbf{T})^2$ since $\boldsymbol{\lambda 2}\mathbf{T}$ contains primitive recursors $\boldsymbol{R}_\sigma$ for *all* types $\sigma \in \mathbb{T}_2$. As a consequence, extending $\lambda\mathbf{T}$ to $\boldsymbol{\lambda 2}\mathbf{T}$ increases the computation power on the first-order numerals $\bar{0}, \bar{1}, \bar{2}, \ldots$ of type $\mathbf{0}$. This can be seen as follows. Write $\boldsymbol{c}_n \equiv \Lambda\alpha\lambda f^{\alpha\to\alpha} x^\alpha.f^n(x)$ for the n-th *polymorphic Church numeral* of type $\mathbf{PolyNat} \equiv \forall\alpha.(\alpha\to\alpha)\to\alpha\to\alpha$.

3.1.9. THEOREM. (i) *The $\lambda\mathbf{T}$-representable functions on the first-order numerals are exactly those that are provably total in Peano Arithmetic* ($\mathbf{PA}$).

(ii) *The $\boldsymbol{\lambda 2}$-representable functions on the polymorphic Church numerals are exactly the provably total functions of second-order Heyting arithmetic* ($\mathbf{HA_2}$).

PROOF. (i) See Gödel (1958).

(ii) See Girard (1972). In Fortune et al. (1983), some examples of rapidly growing definable functions (e.g., ε_0 recursive) are given. □

Using the 'new' primitive recursors in $\boldsymbol{\lambda 2}\mathbf{T}$ one can transfer the computation power on $\mathbf{PolyNat}$ to $\mathbf{0}$.

3.1.10. PROPOSITION. *In $\boldsymbol{\lambda 2}\mathbf{T}$ there exist terms*

$$\begin{aligned} \mathbf{T}_1 \quad &\textit{of type} \quad \mathbf{PolyNat}\to\mathbf{0}, \\ \mathbf{T}_2 \quad &\textit{of type} \quad \mathbf{0}\to\mathbf{PolyNat}, \end{aligned}$$

such that for each $n \in \mathbb{N}$

$$\begin{aligned} \mathbf{T}_1\boldsymbol{c}_n &= \bar{n}, \\ \mathbf{T}_2\bar{n} &= \boldsymbol{c}_n. \end{aligned}$$

PROOF. Take

$$\mathbf{T}_1 \equiv \lambda x^{\mathbf{PolyNat}}.x\,\mathbf{0}\boldsymbol{S}\mathbf{0},$$

and

$$\begin{aligned} \mathbf{T}_2 &\equiv \lambda x^0.\boldsymbol{R}_{\mathbf{PolyNat}}\boldsymbol{c}_0(\lambda y^0 z^{\mathbf{PolyNat}}.\mathbf{Succ}\,z)x \\ &= \boldsymbol{R}_{\mathbf{PolyNat}}\boldsymbol{c}_0(\lambda y^0.\mathbf{Succ}), \end{aligned}$$

where $\mathbf{Succ} \equiv \lambda m^{\mathbf{PolyNat}}\Lambda\alpha\lambda f^{\alpha\to\alpha} x^\alpha.f(m\alpha f x)$ represents the successor function on the polymorphic Church numerals. □

3.1.11. COROLLARY. *The* $\boldsymbol{\lambda 2T}$*-definable functions on the first-order numerals are those that are provably total in* $\mathbf{HA_2}$.

Plain second-order extensions are investigated in Breazu-Tannen and Meyer (1987). We will return to this in Section 3.6.

3.2. Semantics of First-Order Systems

In this section we describe the basic semantical notions for $\boldsymbol{\lambda}^{\tau}$ and its extensions. The exposition is based on work by Friedman (1975). The mathematical basis of first-order semantics is the notion of type structure (see Section 2.1).

Below, $\boldsymbol{\lambda}\Box$ ranges over the first order systems.

3.2.1. DEFINITION. Let $\mathfrak{M}$ be a type structure.

(i) A (term) *valuation* in $\mathfrak{M}$ is a map

$$\rho : \mathrm{Var}(\boldsymbol{\lambda}\Box) \to \mathfrak{M}$$

such that for all variables x^{σ}

$$\rho(x^{\sigma}) \in \mathfrak{M}_{\sigma}.$$

The set of all valuations is denoted by $\mathrm{Val}(\mathfrak{M})$. In the sequel, $\rho, \rho', \ldots$ range over valuations.

(ii) A $\boldsymbol{\lambda}\Box$*-interpretation* in $\mathfrak{M}$ is a map

$$[\![\ ]\!] : \mathrm{Term}(\boldsymbol{\lambda}\Box) \times \mathrm{Val}(\mathfrak{M}) \to \mathfrak{M}$$

(we write $[\![M]\!]_{\rho}$ instead of $[\![\ ]\!](M, \rho)$) that is *type correct*, i.e. for all M, ρ

$$M \in \mathrm{Term}_{\sigma}(\boldsymbol{\lambda}\Box) \ \Rightarrow\ [\![M]\!]_{\rho} \in \mathfrak{M}_{\sigma},$$

and moreover satisfies

$$\begin{aligned} [\![x]\!]_{\rho} &= \rho(x), \\ [\![MN]\!]_{\rho} &= [\![M]\!]_{\rho} \cdot [\![N]\!]_{\rho}, \\ [\![\lambda x^{\sigma}.M]\!]_{\rho} \cdot a &= [\![M]\!]_{\rho(x\,=a)} \quad \text{for each } a \in \mathfrak{M}_{\sigma}, \end{aligned}$$

and

$$\rho \restriction \mathrm{FV}(M) = \rho' \restriction \mathrm{FV}(M) \ \Rightarrow\ [\![M]\!]_{\rho} = [\![M]\!]_{\rho'}.$$

3.2 2. DEFINITION. A $\boldsymbol{\lambda}\Box$*-structure* consists of a type structure $\mathfrak{M}$ together with a $\boldsymbol{\lambda}\Box$-interpretation in $\mathfrak{M}$:

$$\mathfrak{M} = \langle (\mathfrak{M}_{\sigma})_{\sigma \in \mathbb{T}},\ (\mathrm{App}_{\sigma,\tau})_{\sigma,\tau \in \mathbb{T}},\ [\![\ ]\!] \rangle.$$

Notions for type structures, such as 'extensionality' and 'ω-structure', carry over to $\boldsymbol{\lambda}\Box$-structures in the obvious way.

Recall the full type structures $\mathfrak{M}(X)$ There is a straightforward way to interpret $\boldsymbol{\lambda}^{\tau}$ terms in this structure, namely as functionals In the $\mathfrak{M}(\mathbb{N})$ case this interpretation can be extended to $\boldsymbol{\lambda}\mathbf{T}$

3 2 3 DEFINITION (i) The $\boldsymbol{\lambda}^{\tau}$-interpretation $[\![\ ]\!]$ in $\mathfrak{M}(X)$ is defined by

$$\begin{aligned} [\![x]\!]_{\rho} &= \rho(x), \\ [\![MN]\!]_{\rho} &= [\![M]\!]_{\rho}([\![N]\!]_{\rho}), \\ [\![\lambda x^{\sigma}\, M]\!]_{\rho} &= \boldsymbol{\lambda} a{\in}X_{\sigma}\, [\![M]\!]_{\rho(x\,=a)}, \end{aligned}$$

where $\boldsymbol{\lambda}$ denotes meta lambda abstraction One easily verifies that $[\![\ ]\!]$ satisfies the requirements in Definition 3 2 1 This gives a $\boldsymbol{\lambda}^{\tau}$-structure, which is also denoted by $\mathfrak{M}(X)$

(ii) Consider $\mathfrak{M}(\mathbb{N})$ Extend the interpretation in (i) with

$$\begin{aligned} [\![\boldsymbol{0}]\!]_{\rho} &= 0, \\ [\![\boldsymbol{S}]\!]_{\rho} &= \boldsymbol{\lambda} n{\in}\mathbb{N}\; n{+}1, \\ [\![\boldsymbol{R}_{\sigma}]\!]_{\rho} &= f_{\sigma}, \end{aligned}$$

where f_{σ} is such that

$$\begin{aligned} f_{\sigma} abn &= a && \text{if } n = 0, \\ &= b(n{-}1)(f_{\sigma} ab(n{-}1)) && \text{if } n > 0 \end{aligned}$$

(Note that such an f_{σ} exists in $\mathbb{N}_{\sigma\to(\mathbf{0}\to\sigma\to\sigma)\to\mathbf{0}\to\sigma}$) This gives a $\boldsymbol{\lambda}\mathbf{T}$-interpretation in $\mathfrak{M}(\mathbb{N})$

The interpretation of types ($[\![\sigma]\!] = \mathfrak{M}_{\sigma}$) is not mentioned explicitly The notion of *satisfaction* of equations and sequents is defined in the obvious way

3 2 4 DEFINITION Let $\mathfrak{M}$ be a $\boldsymbol{\lambda}\Box$-structure as above

(i) For $M, N \in \mathrm{Term}_{\sigma}(\boldsymbol{\lambda}\Box)$ one defines

$$\begin{aligned} \mathfrak{M}, \rho \models M = N \quad &\Leftrightarrow \quad [\![M]\!]_{\rho} = [\![N]\!]_{\rho}, \\ \mathfrak{M} \models M = N \quad &\Leftrightarrow \quad \text{for all } \rho \quad \mathfrak{M}, \rho \models M = N \end{aligned}$$

(ii) $\mathfrak{M}$ *satisfies* $\Gamma \vdash E$ (notation $\Gamma \models_{\mathfrak{M}} E$) if for all ρ

$$\mathfrak{M}, \rho \models E \quad \text{whenever for each } A \text{ in } \Gamma \quad \mathfrak{M}, \rho \models A$$

(iii) $\mathfrak{M}$ is a *model of* $\boldsymbol{\lambda}\Box$ if

$$\Gamma \vdash_{\boldsymbol{\lambda}\Box} E \;\Rightarrow\; \Gamma \models_{\mathfrak{M}} E$$

Now we can show that the $\boldsymbol{\lambda}^{\tau}$ structures $\mathfrak{M}(X)$ are in fact models of $\boldsymbol{\lambda}^{\tau}$, and likewise $\mathfrak{M}(\mathbb{N})$ for $\boldsymbol{\lambda}\mathbf{T}$

We first state and prove some general technical results.

3.2.5. SUBSTITUTION LEMMA. *Let $\mathfrak{M}$ be a $\boldsymbol{\lambda}\Box$-structure. Suppose $\mathfrak{M}$ is extensional. Let $M, N \in \mathrm{Term}(\boldsymbol{\lambda}\Box)$ with $N \in \mathrm{Term}_{\sigma}(\boldsymbol{\lambda}\Box)$. Then*

$$[\![M[x^{\sigma} := N]]\!]_{\rho} = [\![M]\!]_{\rho(x^{\sigma} := [\![N]\!]_{\rho})}$$

PROOF. Induction on M. The cases $M \equiv x^{\sigma}$, $M \equiv y^{\tau}$, $M \equiv c$, $M \equiv M_1 M_2$ are easy. As to the case $M \equiv \lambda y^{\tau}.M_1$, let $a \in \mathfrak{M}_{\sigma}$. Then

$$\begin{aligned} [\![(\lambda y^{\tau}.M_1)[x := N]]\!]_{\rho} \cdot a &= [\![M_1[x := N]]\!]_{\rho(y := a)} \\ &= [\![M_1]\!]_{\rho(y := a, x := [\![N]\!]_{\rho})}, \quad \text{by induction hypothesis} \\ &= [\![\lambda y^{\tau}.M_1]\!]_{\rho(x := [\![N]\!]_{\rho})} \cdot a. \ \Box \end{aligned}$$

3.2 6. REMARK. In fact, the condition that $\mathfrak{M}$ is extensional can be relaxed. As to a more refined approach, call $\mathfrak{M}$ a *ξ-structure* if $\mathfrak{M}$ satisfies the ξ-axiom:

$$\mathfrak{M} \models \forall x^{\sigma}\ M = N \to \lambda x^{\sigma}.M = \lambda x^{\sigma}.N,$$

i.e.

$$\forall a \in \mathfrak{M}_{\sigma}\ \ [[\![M]\!]_{\rho(x^{\sigma} := a)} = [\![N]\!]_{\rho(x^{\sigma} := a)}] \ \Rightarrow\ [\![\lambda x^{\sigma}.M]\!]_{\rho} = [\![\lambda x^{\sigma}.N]\!]_{\rho}.$$

One can prove the above substitution result for ξ-structures (however with considerable effort like in the untyped case, cf. Barendregt (1984), Lemma 5.3.3). But one also has

$$\mathfrak{M} \text{ is extensional } \Rightarrow\ \mathfrak{M} \text{ is a } \xi\text{-structure}$$

(see also Barendsen (1990)). Since our models are extensional we work with extensionality to shorten our proofs.

3.2.7. THEOREM. *Let $\mathfrak{M}$ be extensional. Then $\mathfrak{M}$ is a model of $\boldsymbol{\lambda}^{\tau}$.*

PROOF. By induction on the derivation of $\Gamma \vdash_{\boldsymbol{\lambda}^{\tau}} M = N$. As to the rules and axioms of (L1), for (β_1) note that

$$\begin{aligned} [\![(\lambda x^{\sigma}.M)N]\!]_{\rho} &= [\![\lambda x^{\sigma}.M]\!]_{\rho} \cdot [\![N]\!]_{\rho} \\ &= [\![M]\!]_{\rho(x^{\sigma} := [\![N]\!]_{\rho})} \\ &= [\![M[x^{\sigma} := N]]\!]_{\rho}, \quad \text{by the Substitution Lemma.} \end{aligned}$$

The soundness of the (η_1)-axiom is verified as follows. For each $a \in \mathfrak{M}_{\sigma}$

$$\begin{aligned} [\![\lambda x^{\sigma}.Mx]\!]_{\rho} \cdot a &= [\![Mx]\!]_{\rho(x^{\sigma} := a)} \\ &= [\![M]\!]_{\rho} \cdot [\![x]\!]_{\rho(x^{\sigma} := a)}, \quad \text{since } x \notin \mathrm{FV}(M) \\ &= [\![M]\!]_{\rho} \cdot a. \end{aligned}$$

Hence by extensionality $[\![\lambda x^{\sigma}.Mx]\!]_{\rho} = [\![M]\!]_{\rho}$. The compatibility rules are easily checked. $\Box$

3.2.8. COROLLARY. (i) $\mathfrak{M}(X)$ *is a model of* $\boldsymbol{\lambda}^{\tau}$.
(ii) $\mathfrak{M}(\mathbb{N})$ *is a model of* $\boldsymbol{\lambda}\mathbf{T}$.

PROOF. (i) By Theorem 3.2.7.
(ii) By extension of the proof of Theorem 3.2.7. The axioms in (CR) are obviously satisfied by construction of $[\![\boldsymbol{R}_\sigma]\!]$, $[\![\boldsymbol{S}]\!]$ and $[\![\boldsymbol{0}]\!]$. □

3.3. Semantics of Second-Order Systems

In this subsection we will describe a generalized version of a well-known model construction for some second-order systems. A general setting for arbitrary second-order models will not be given; the reader is referred to Bruce et al. (1990) and to Breazu-Tannen and Coquand (1988).

A straightforward function-theoretic construction does not work because of the impredicative nature of second-order λ-calculus. E.g., the term $\mathsf{I} \equiv \Lambda\alpha.\lambda x^{\alpha}.x$ of type $\forall\alpha.\alpha{\to}\alpha$ can be applied to its own type. Sticking to the function-theoretic setting one is tempted to interpret I as a function. But then $[\![\mathsf{I}]\!] \in [\![\forall\alpha.\alpha{\to}\alpha]\!]$, whereas at the same time $[\![\forall\alpha.\alpha{\to}\alpha]\!]$ must be a valid argument for $[\![\mathsf{I}]\!]$, which is impossible by the foundation axiom in set theory.

The model construction described below makes use of the idea of partial equivalence relations on the domain of a certain (untyped) applicative structure.

Since the types may now contain free variables, the type interpretation is no longer fixed but depends on a type valuation dealing with type variables.

The interpretation of terms is very simple: first the type information is erased and then the resulting term is interpreted in the untyped structure mentioned above.

3.3.1. DEFINITION. (i) A *partial applicative structure* is a structure

$$\mathfrak{A} = \langle A, \cdot \rangle$$

where A is a non-empty set, called the *domain* of $\mathfrak{A}$, and $\cdot : A \times A \rightharpoonup A$ is a *partial application function*. Again we often write ab for $a \cdot b$.

(ii) By $\simeq$ we denote *partial equality* of expressions: $p \simeq q$ iff either p and q are undefined, or p and q are both defined and equal.

Given a partial applicative structure, we consider the collection of so-called partial equivalence relations on its domain. The second-order types are then interpreted as elements of this collection.

3.3.2. DEFINITION. Let A be a set, and $R \subseteq A \times A$.
(i) R is a *partial equivalence relation* (*per*) if R is symmetric and transitive. By $\mathrm{PER}(A)$ we denote the collection of all such relations.

(ii) The *domain* of R is the set

$$\operatorname{dom} R = \{a \in A \mid \exists a' \in A \ [a \ R \ a' \text{ or } a' \ R \ a]\}.$$

3.3.3. LEMMA. *Let R be a per on A. Then*

$$\operatorname{dom} R = \{a \in A \mid a \ R \ a\}.$$

PROOF. ($\subseteq$) Let $a \in \operatorname{dom} R$, say (without loss of generality) $a \ R \ a'$. Then $a' \ R \ a$ by symmetry, so $a \ R \ a$ by transitivity.
($\supseteq$) Trivial. □

Note that a per R is an equivalence relation on $\operatorname{dom} R$.
For the following definitions, fix a partial applicative structure $\mathfrak{A} = \langle A, \cdot \rangle$.

3.3.4. DEFINITION (i) For relations R, S on A, the *function relation* $R{\to}S$ is defined as follows.

$$a \ R{\to}S \ a' \ \Leftrightarrow \ \forall b, b' \ [b \ R \ b' \ \Rightarrow \ ab \ S \ a'b'],$$

where it is understood that $ab \ S \ a'b'$ only holds if both $ab{\downarrow}$ and $a'b'{\downarrow}$.
(ii) If $(R_i)_{i \in I}$ is a collection of relations on A, then the *intersection relation* $\bigwedge_{i \in I} R_i$ is defined by setting

$$a \bigwedge_{i \in I} R_i \ a' \ \Leftrightarrow \ \forall i \in I \ [a \ R_i \ a'].$$

3.3.5. LEMMA. (i) *If $R, S \in \mathrm{PER}(A)$, then $R{\to}S \in \mathrm{PER}(A)$.*
(ii) *If $(R_i)_{i \in I}$ is a collection in $\mathrm{PER}(A)$, then $\bigwedge_{i \in I} R_i \in \mathrm{PER}(A)$*

PROOF. (i) Suppose $R, S \in \mathrm{PER}(A)$. The symmetry of $R{\to}S$ follows directly from the symmetry of R and S. As to transitivity, let $a, a', a'' \in A$. Suppose

$$a \ R{\to}S \ a', \quad a' \ R{\to}S \ a''.$$

Let $b, b' \in A$ with $b \ R \ b'$. Then $ab \ S \ a'b'$. Moreover $b' \ R \ b'$ by Lemma 3.3.3, so $a'b' \ S \ a''b'$. Therefore by transitivity of S one has $ab \ S \ a''b$. Hence $a \ R{\to}S \ a''$.
(ii) Straightforward. □

If we have an interpretation of the type constants in $\mathrm{PER}(A)$, then each polymorphic type can be mapped into $\mathrm{PER}(A)$. The resulting structure is called a per structure (cf. 'type structure' in the first order case).

3 3.6. DEFINITION. (i) A *per structure* is a structure

$$\mathfrak{P} = \langle A, \cdot, \mathcal{T} \rangle$$

consisting of a partial applicative structure and a valuation of type constants $\mathcal{T} : \mathbb{C} \to \mathrm{PER}(A)$.

(ii) For each $\sigma \in \mathbb{T}_2$ and type valuation $\xi : \mathbb{V} \to \mathrm{PER}(A)$ the *per interpretation* in $\mathfrak{P}$ of σ under ξ, notation $[\sigma]_\xi = [\sigma]^{\mathcal{T}}_\xi \in \mathrm{PER}(A)$, is defined inductively as follows.

$$\begin{aligned} [c]_\xi &= \mathcal{T}(c), \\ [\alpha]_\xi &= \xi(\alpha), \\ [\sigma{\to}\tau]_\xi &= [\sigma]_\xi{\to}[\tau]_\xi, \\ [\forall\alpha.\sigma]_\xi &= \bigwedge_{R\in\mathrm{PER}(A)} [\sigma]_{\xi(\alpha:=R)}. \end{aligned}$$

This is a sound definition by Lemma 3.3.5.

Since we allow type systems with constants we also consider untyped λ-terms with constants. If C is a given set then we write $\Lambda(C)$ for the set of untyped terms extended with constants from C.

3.3.7. DEFINITION. A *partial* $\lambda(C)$*-interpretation* in a partial applicative structure $\langle A, \cdot\rangle$ consists of a constant valuation

$$\mathcal{V} : C \to A$$

and a partial map

$$(\![\ ]\!) : \Lambda(C) \times \mathrm{Val}(A) \rightharpoonup A$$

(where $\mathrm{Val}(A)$ is defined in the obvious way), such that

$$\begin{aligned} (\![x]\!)_\rho &\simeq \rho(x), \\ (\![c]\!)_\rho &\simeq \mathcal{V}(c) \quad \text{if } c \in C, \\ (\![MN]\!)_\rho &\simeq (\![M]\!)_\rho \cdot (\![N]\!)_\rho, \\ (\![\lambda x.M]\!)_\rho &\downarrow, \\ (\![\lambda x.M]\!)_\rho \cdot a &\simeq (\![M]\!)_{\rho(x:=a)}, \end{aligned}$$

and moreover

$$\rho \restriction \mathrm{FV}(M) = \rho' \restriction \mathrm{FV}(M) \;\Rightarrow\; (\![M]\!)_\rho \simeq (\![M]\!)_{\rho'}.$$

3.3.8. FACT. There exists a partial λ-interpretation into $\langle A, \cdot\rangle$ if there exist $k, s \in A$ such that for all $a, b, c \in A$

$$\begin{aligned} ka\downarrow, \quad kab &\simeq a, \\ sa\downarrow, \quad sab\downarrow, \quad sabc &\simeq ac(bc). \end{aligned}$$

Then $(\![\ ]\!)$ is obtained by translating each λ-term to its combinatory variant (using $\boldsymbol{K}, \boldsymbol{S}$) and then using k, s above for the interpretation in A.

An example is Kleene's applicative structure $\langle \mathbb{N}, \cdot \rangle$ with a recursion theoretic interpretation of Λ $(= \Lambda(\emptyset))$.

3.3 9. EXAMPLE. Set $\mathcal{K} = \langle \mathbb{N}, \cdot \rangle$ where

$$e \cdot x \simeq \{e\}(x).$$

Define $(\![\ ]\!)$ by

$$\begin{aligned} (\![x]\!)_\rho &= \rho(x), \\ (\![MN]\!)_\rho &\simeq (\![M]\!)_\rho \cdot (\![N]\!)_\rho, \\ (\![\lambda x.M]\!)_\rho &= \tilde{\lambda} n.(\![M]\!)_{\rho(x\,=n)}, \end{aligned}$$

where $\tilde{\lambda} n.\psi(n)$ stands for the choice of a (canonical) index for the partial recursive function ψ. One easily verifies that $\boldsymbol{\lambda}\vec{n}.(\![M]\!)_{\rho(\vec{x}\,=\vec{n})}$ is partial recursive and indices can be found effectively. Then $(\![\ ]\!)$ is a partial λ-interpretation in $\mathcal{K}$.

Now let $\boldsymbol{\lambda}\square$ range over the second-order systems.

3.3.10. DEFINITION. A *per structure for* $\boldsymbol{\lambda}\square$ is a structure

$$\mathfrak{P} = \langle A, \cdot, \mathcal{T}, \mathcal{V}, (\![\ ]\!) \rangle$$

such that

(1) $\langle A, \cdot, \mathcal{T} \rangle$ is a per structure;

(2) $\mathcal{V}$ and $(\![\ ]\!)$ constitute a partial $\lambda(\mathrm{Cons}(\boldsymbol{\lambda}\square))$-interpretation in $\langle A, \cdot \rangle$;

(3) $(\mathcal{T}, \mathcal{V})$ is a *constant valuation pair*, i.e. for each $c \in \mathrm{Cons}_\sigma(\boldsymbol{\lambda}\square)$ and ξ one has $\mathcal{V}(c) \in \mathrm{dom}[\sigma]^{\mathcal{T}}_{\xi}$.

We omit $\mathcal{V}$ if $\mathrm{Cons}(\boldsymbol{\lambda}\square) = \emptyset$.

Terms of $\boldsymbol{\lambda}\square$ are first interpreted in the domain of a per structure $\mathfrak{P}$ by erasing type information and using the interpretation function in $\mathfrak{P}$.

3.3.11. DEFINITION. (i) The *erase-type map* $|\ | : \mathrm{Term}(\boldsymbol{\lambda}\square) \to \Lambda(\mathrm{Cons}(\boldsymbol{\lambda}\square))$ is defined inductively as follows.

$$\begin{aligned} |c| &\equiv c, \\ |x^\sigma| &\equiv x_\sigma, \\ |MN| &\equiv |M||N|, \\ |\lambda x^\sigma\, M| &\equiv \lambda x_\sigma.|M|, \\ |M\sigma| &\equiv |M|, \\ |\Lambda\alpha.M| &\equiv |M|, \end{aligned}$$

where x_σ is an untyped variable uniquely determined by x^σ.

(ii) For $M \in \mathrm{Term}(\lambda\Box)$, $\rho : \mathrm{Var}(\lambda\Box) \to A$ one defines the *per interpretation* of M by

$$[M]_\rho = (\!|\,|M|\,|\!)_{|\rho|}$$

where $|\rho|$ is such that for each variable x^σ

$$|\rho|(|x^\sigma|) = \rho(x^\sigma).$$

(iii) Let $M \in \mathrm{Term}(\lambda\Box)$. Then (ρ, ξ) is a *valuation pair* for M (notation $(\rho, \xi) \succ M$) if for all $x^\sigma \in \mathrm{FV}(M)$ one has $\rho(x^\sigma) \in \mathrm{dom}[\sigma]_\xi$.

Now we want to show that this gives rise to a type correct interpretation, i.e. each per structure is at least a model of type assignment.

3.3.12. DEFINITION. Let $\mathfrak{P}$ be a per structure for $\lambda\Box$.

(i) Let ρ, ρ' be term valuations in $\mathfrak{P}$, and let ξ be a type valuation. Let $X \subseteq \mathrm{Var}(\lambda\Box)$. Then ρ and ρ' are *ξ-equivalent with respect to X* (notation $\rho \approx_\xi^X \rho'$) if

$$\rho(x^\sigma)\ [\sigma]_\xi\ \rho'(x^\sigma) \quad \text{for all } x^\sigma \in X.$$

(ii) By $\rho \approx_\xi^M \rho'$ we abbreviate $\rho \approx_\xi^{\mathrm{FV}(M)} \rho'$.

Note that if $\rho \approx_\xi^M \rho'$ then both (ρ, ξ) and (ρ', ξ) are valuation pairs for M

3.3.13 SUBSTITUTION LEMMA. $[\sigma[\alpha := \tau]]_\xi = [\sigma]_{\xi(\alpha := [\tau]_\xi)}$.

PROOF. Straightforward induction on the structure of σ. □

The main technical tool to show type correctness of the per interpretation of terms is the following.

3.3.14. PROPOSITION. *For all $M \in \mathrm{Term}_\sigma(\lambda\Box)$ and all valuations ρ, ρ', ξ*

$$\rho \approx_\xi^M \rho' \ \Rightarrow\ [M]_\rho\ [\sigma]_\xi\ [M]_{\rho'}.$$

PROOF. Induction on M.

- $M \equiv c$. Then the result holds since $(\mathcal{T}, \mathcal{V})$ is a constant valuation pair
- $M \equiv x^\sigma$. Then by assumption $[x^\sigma]_\rho\ [\sigma]_\xi\ [x^\sigma]_{\rho'}$.
- $M \equiv M_1 M_2$. By induction hypothesis we can assume

$$\begin{array}{lll} [M_1]_\rho & [\tau \to \sigma]_\xi & [M_1]_{\rho'}, \\ [M_2]_\rho & [\tau]_\xi & [M_2]_{\rho'}. \end{array}$$

Then by definition of $[\tau \to \sigma]_\xi$ one has

$$[M_1]_\rho \cdot [M_2]_\rho\ [\sigma]_\xi\ [M_1]_{\rho'} \cdot [M_2]_{\rho'},$$

and we are done since $[M_1]_\rho \cdot [M_2]_\rho = [M_1 M_2]_\rho$.

• $M \equiv \lambda x^{\sigma}.M_1$ where $M_1 \in \mathrm{Term}_{\tau}(\boldsymbol{\lambda}\Box)$. Suppose $a[\sigma]_{\xi}a'$. Then $\rho(x := a) \approx_{\xi}^{M_1} \rho'(x := a')$, so by induction hypothesis

$$[M_1]_{\rho(x^{\sigma}=a)}\,[\tau]_{\xi}\,[M_1]_{\rho'(x^{\sigma}=a')}.$$

Therefore by the properties of the interpretation $(\![\ ]\!)$

$$[\lambda x^{\sigma}.M]_{\rho}\,[\sigma\to\tau]_{\xi}\,[\lambda x^{\sigma}.M]_{\rho'}.$$

• $M \equiv M_1\tau$ where $M_1 \in \mathrm{Term}_{\forall\alpha\,\sigma}(\boldsymbol{\lambda}\Box)$. By induction hypothesis one has

$$[M_1]_{\rho} \bigwedge_{R\in\mathrm{PER}(A)} [\sigma]_{\xi(\alpha=R)}\,[M_1]_{\rho'},$$

so

$$[M_1]_{\rho}\,[\sigma]_{\xi(\alpha=[\tau]_{\xi})}\,[M_1]_{\rho'}.$$

Now by the Substitution Lemma 3.3.13 and the fact that $[M_1\tau]_{\rho} = [M_1]_{\rho}$ the result follows.

• $M \equiv \Lambda\alpha.M_1$ where $M_1 \in \mathrm{Term}_{\sigma}(\boldsymbol{\lambda}\Box)$. Let $R \in \mathrm{PER}(A)$ Then $\rho \approx_{\xi(\alpha=R)}^{M_1} \rho'$ by the restriction on occurrences of α. Then by induction hypothesis

$$[M_1]_{\rho}\,[\sigma_1]_{\xi(\alpha=R)}\,[M_1]_{\rho'}.$$

So we can conclude

$$[M_1]_{\rho} \bigwedge_{R\in\mathrm{PER}(A)} [\sigma_1]_{\xi(\alpha=R)}\,[M_1]_{\rho'},$$

and we are done since $[M_1]_{\rho} = [\Lambda\alpha.M_1]_{\rho}$. □

3.3.15. COROLLARY (Type Correctness). *Let* $M \in \mathrm{Term}_{\sigma}(\boldsymbol{\lambda}\Box)$. *Then*

$$(\rho,\xi) \succ M \;\Rightarrow\; [M]_{\rho} \in \mathrm{dom}[\sigma]_{\xi}.$$

PROOF. Suppose $(\rho,\xi) \succ M$. Then obviously $\rho \approx_{\xi}^{M} \rho$, so by Proposition 3.3.14

$$[M]_{\rho}[\sigma]_{\xi}[M]_{\rho}.\ \Box$$

Now a proper interpretation of types and terms in a per structure $\mathfrak{P}$ for $\boldsymbol{\lambda}\Box$, depending on a valuation pair (ρ,ξ), can be given.

3.3.16. DEFINITION. (i) Let $\sigma \in \mathbb{T}_2$. The *interpretation* of σ in $\mathfrak{P}$ under type valuation ξ is

$$[\![\sigma]\!]_{\xi} = {}^{\mathrm{dom}[\sigma]_{\xi}}\!\Big/_{[\sigma]_{\xi}}.$$

(ii) For $M \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box)$ and $(\rho, \xi) \succ M$

$$[\![M]\!]_{\rho,\xi} = [M]_\rho \pmod{[\sigma]_\xi}.$$

By Corollary 3.3.15 this is a sound definition.

We conclude the analysis as follows.

3.3.17. THEOREM. *$\mathfrak{P}$ is a model of type assignment, i.e. for all $M \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box)$ and $(\rho, \xi) \succ M$*

$$[\![M]\!]_{\rho,\xi} \in [\![\sigma]\!]_\xi.$$

3.3.18. DEFINITION. The notion of *validity* in $\mathfrak{P}$ is defined in the usual way. Moreover $\mathfrak{P}$ is said to be a *model of* $\boldsymbol{\lambda}\Box$ if

$$\Gamma \vdash_{\boldsymbol{\lambda}\Box} E \;\Rightarrow\; \Gamma \models_{\mathfrak{P}} E.$$

Disregarding the constants, this construction automatically gives a model of $\boldsymbol{\lambda 2}$. This will now be shown.

3.3.19. SUBSTITUTION LEMMA. *For all M, N and appropriate ρ, ξ*

$$[\![M[x^\sigma := N]]\!]_{\rho,\xi} = [\![M]\!]_{\rho(x^\sigma := [\![N]\!]_{\rho,\xi}),\xi}.$$

PROOF. By induction on the structure of $M \in \mathrm{Term}_\sigma(\boldsymbol{\lambda}\Box)$ we show

$$[M[x^\sigma := N]]_\rho\ [\sigma]_\xi\ [M]_{\rho(x^\sigma := [N]_\rho)}.$$

The cases $M \equiv x$, $M \equiv y$, $M \equiv M_1 M_2$, $M \equiv M_1 \tau$, $M \equiv \Lambda\alpha.M_1$ are easy to handle. As to $M \equiv \lambda y^\tau.M_1$ with $\sigma \equiv \tau \to \sigma_1$, take a, a' with $a\ [\tau]_\xi\ a'$. Then by the induction hypothesis and Proposition 3.3.14

$$[M_1[x^\sigma := N]]_{\rho(y=a)}\ [\sigma_1]_\xi\ [M_1]_{\rho(x^\sigma := [N]_\rho, y^\tau = a')}$$

so

$$[\lambda y^\tau.M_1[x^\sigma := N]]_\rho \cdot a\ [\sigma_1]_\xi\ [M_1]_{\rho(x^\sigma := [N]_\rho)} \cdot a'$$

which gives us the desired result. □

Note the similarity with the proof of the Substitution Lemma in the first-order case (3.2.5). In the present case, the built-in extensionality of the per construction is used.

3.3.20. PROPOSITION. *Let $\mathfrak{P}$ be a per structure for $\boldsymbol{\lambda}\Box$. Then the axioms and rules in* (L1) *and* (L2) *are valid in $\mathfrak{P}$.*

PROOF We only treat the principal cases, e g , the compatibility rules are easily verified

As to (β_1), note that for each $M \in \mathrm{Term}_\tau(\boldsymbol{\lambda}\Box)$ and each valuation pair (ρ, ξ) for M

$$\begin{aligned}[(\lambda x^\sigma\ M)N]_\rho &= [\lambda x^\sigma\ M]_\rho\ [N]_\rho \\ &= [M]_{\rho(x^\sigma := [N]_\rho)} \\ [\tau]_\xi\ &[M[x^\sigma := N]]_\rho, \quad \text{by the Substitution Lemma 3 3 19}\end{aligned}$$

For (η_1), let $M \in \mathrm{Term}_{\sigma\to\tau}(\boldsymbol{\lambda}\Box)$ Let $a, a' \in A$ such that $a\ [\sigma]_\xi\ a'$ Then

$$\begin{aligned}[\lambda x^\sigma\ Mx]_\rho\ a &= [Mx]_{\rho(x^\sigma := a)} \\ &= [M]_\rho\ [x]_{\rho(x^\sigma := a)} \\ [\tau]_\xi\ &[M]_\rho\ [x]_{\rho(x^\sigma := a')}, \quad \text{by Proposition 3 3 14} \\ &= [M]_\rho \cdot a'\end{aligned}$$

Hence $[\lambda x^\sigma\ Mx]_\rho\ [\sigma\to\tau]_\xi\ [M]_\rho$

The rules of (L2) are trivially sound by the fact that the type information is erased in the interpretation process $\Box$

3 3 21 COROLLARY *Let $\mathfrak{P}$ be a per structure for $\boldsymbol{\lambda 2}$ Then $\mathfrak{P}$ is a model of $\boldsymbol{\lambda 2}$*

So the lambda calculus axioms and rules are 'automatically' satisfied in a per structure This shows that the question whether a per structure $\mathfrak{P}$ is a model of $\boldsymbol{\lambda}\Box$ depends entirely on the constant interpretations $\mathcal{T}, \mathcal{V}$

The structure $\mathcal{K}$ (see Example 3 3 9) can be extended to a model of $\boldsymbol{\lambda 2T}$

3 3 22 DEFINITION (i) Using the recursion theorem, determine $r \in \mathbb{N}$ such that in $\mathcal{K}$

$$\begin{aligned}rabn &\simeq a && \text{if } n = 0, \\ &\simeq b(n{-}1)(rab(n{-}1)) && \text{if } n > 0\end{aligned}$$

(ii) The per structure HEO2 is defined by

$$\mathrm{HEO2} = \langle \mathbb{N},\ ,\ \mathcal{T},\ \mathcal{V},\ (\![\ ]\!) \rangle$$

where

$$\mathcal{T}(\mathbf{0}) = \mathrm{EQ}_{\mathbb{N}} = \{(n, n) \mid n \in \mathbb{N}\}$$

and

$$\begin{aligned}\mathcal{V}(\boldsymbol{0}) &= 0, \\ \mathcal{V}(\boldsymbol{S}) &= \tilde{\lambda} n\ n{+}1, \\ \mathcal{V}(\boldsymbol{R}_\sigma) &= r\end{aligned}$$

3 3 23 PROPOSITION HEO2 *is a model of* $\boldsymbol{\lambda 2T}$

PROOF By induction on the derivation showing $\Gamma \vdash_{\boldsymbol{\lambda 2T}} \varphi$ one shows the validity of each provable sequent The rules and axioms in (L1) and (L2) are valid by Proposition 3 3 20 Furthermore for all appropriate M, N, ρ, ξ

$$\begin{aligned} \mathrm{HEO2}, \rho, \xi &\models \boldsymbol{R}_\sigma MN\boldsymbol{0} =_\sigma M, \\ \mathrm{HEO2}, \rho, \xi &\models \boldsymbol{R}_\sigma MN(\boldsymbol{S}P) =_\sigma NP(\boldsymbol{R}_\sigma MNP), \end{aligned}$$

by the above recursion-theoretic construction □

If one restrict the types to $\mathbb{T}_1$ we obtains a $\boldsymbol{\lambda}^\tau/\boldsymbol{\lambda T}$-structure (in the sense of Definition 3 2 2), known as HEO

3 3 24 COROLLARY *Set*

$$\mathrm{HEO} = \langle ([\![\sigma]\!])_{\sigma\in\mathbb{T}_1},\ (\mathrm{App}_{\sigma\,\tau})_{\sigma,\tau\in\mathbb{T}_1},\ [\![\ ]\!]\rangle$$

where $\mathrm{App}_{\sigma\,\tau}([a],[b]) = [a\ \ b]$, *and* $[\![\ ]\!]$ *is restricted to terms of* $\boldsymbol{\lambda}^\tau/\boldsymbol{\lambda T}$ *Then* HEO *is a model of* $\boldsymbol{\lambda}^\tau$ *and of* $\boldsymbol{\lambda T}$

The structure HEO can be viewed as the first-order fragment of HEO2 In the next section, the relation between first and second order semantical structures will be investigated further

3.4. Building Per Structures over Type Structures

The rest of this chapter concerns the extension of first-order mathematical structures (type structures) to second-order ones (per structures) The results can be used to extend first-order models to models of polymorphic lambda calculi

We proceed as follows We add a type structure to untyped λ-calculus, using the techniques developed in Chapter 2 We consider a per structure based on the applicative theory thus obtained

This technique can be applied to any type structure However, some elements of the type structure in question might get lost in the per construction If the type structure is closed under λ definability (see below) then the elements survive in the per construction, and one obtains a per structure that is in its simple types isomorphic to the original type structure

Let $\mathfrak{M}$ be an extensional ω-structure In this section we will consider $\Lambda^\circ\mathfrak{M}$ (modulo $=_{\beta\mathfrak{M}}$) as an applicative structure Due to the generality of our Church-Rosser result one can also work with the open term model of $\lambda\mathfrak{M}$, but this is not necessary for our purposes

3 4 1 DEFINITION (i) For $M \in \Lambda^{\circ}\mathfrak{M}$ we write the *$\beta\mathfrak{M}$-equivalence class* of M as

$$\langle\!\langle M \rangle\!\rangle = \{N \in \Lambda^{\circ}\mathfrak{M} \mid M =_{\beta\mathfrak{M}} N\}$$

(ii) $\mathcal{L}_{\mathfrak{M}} = \{\langle\!\langle M \rangle\!\rangle \mid M \in \Lambda^{\circ}\mathfrak{M}\}$

(iii) *Application* in $\mathcal{L}_{\mathfrak{M}}$ is defined as usual

$$\langle\!\langle M \rangle\!\rangle \ \langle\!\langle N \rangle\!\rangle = \langle\!\langle MN \rangle\!\rangle$$

Note that this is a sound definition, i e the resulting equivalence class is independent of the choice of the representatives of $\langle\!\langle M \rangle\!\rangle$ and $\langle\!\langle N \rangle\!\rangle$

$\mathcal{L}_{\mathfrak{M}}$ can be extended to a per structure by interpreting $\mathbf{0}$ as a canonical per on the Church numerals

3 4 2 DEFINITION The *per structure* over $\mathfrak{M}$ is defined by

$$\mathfrak{P}_{\mathfrak{M}} = \langle \mathcal{L}_{\mathfrak{M}}, \ , \ \mathcal{T} \rangle$$

where

$$\mathcal{T}(\mathbf{0}) = \{(\langle\!\langle \ulcorner n \urcorner \rangle\!\rangle, \langle\!\langle \ulcorner n \urcorner \rangle\!\rangle) \mid n \in \mathfrak{M}_0\}$$

The rest of this section is devoted to a comparison of $\mathfrak{M}$ and $\mathfrak{P}_{\mathfrak{M}}$

3 4 3 DEFINITION Let $\mathfrak{P} = \langle A, \ , \ \mathcal{T} \rangle$ be a per structure The *first-order fragment* of $\mathfrak{P}$ (notation $\mathfrak{P}^1$) is the type structure

$$\mathfrak{P}^1 = \langle ([\![\sigma]\!])_{\sigma \in \mathbb{T}_1}, \ (\ _{\sigma,\tau})_{\sigma,\tau \in \mathbb{T}_1} \rangle,$$

where $[\![\cdot]\!]$ is the per interpretation of types in $\mathfrak{P}$, and $\ _{\sigma,\tau}$ is defined from $\cdot$ on the quotient spaces $\mathrm{dom}[\sigma]/[\sigma]$ by setting

$$[a]_{[\sigma \to \tau]} \ _{\sigma,\tau} [b]_{[\sigma]} = [a \ \ b]_{[\tau]},$$

where $[x]_R$ denotes the equivalence class of x modulo $R \in \mathrm{PER}(A)$ (provided $x \in \mathrm{dom}\, R$) From the per construction it follows that these application functions are well defined

In order to formulate the main result of this section we introduce some auxiliary terminology

3 4 4 DEFINITION Let $\mathfrak{M}$ be a type structure, and let $\varphi \ \ \mathfrak{M}_{\sigma_1} \times \cdot \times \mathfrak{M}_{\sigma_k} \to \mathfrak{M}_0$ be a map

(i) Let $F \in \Lambda^\circ\mathfrak{M}$. Then F is said to $\lambda\mathfrak{M}$-*define* φ if for all $a_1 \in \mathfrak{M}_{\sigma_1}, \ldots, a_k \in \mathfrak{M}_{\sigma_k}$

$$F\underline{a_1}\cdots\underline{a_k} =_{\beta\mathfrak{M}} \underline{\varphi(a_1,\ldots,a_k)},$$

which becomes in vector denotation

$$F\underline{\vec{a}} =_{\beta\mathfrak{M}} \underline{\varphi(\vec{a})}.$$

(ii) φ is $\lambda\mathfrak{M}$-*definable* if φ is $\lambda\mathfrak{M}$-defined by some $F \in \Lambda^\circ\mathfrak{M}$.

3.4.5. DEFINITION. (i) Let $a \in \mathfrak{M}_{\sigma_1\to\ \to\sigma_k\to\mathbf{0}}$. The *external map* corresponding to a (notation $\hat{a}$) is defined as follows.

$$\hat{a} : \mathfrak{M}_{\sigma_1} \times \cdots \times \mathfrak{M}_{\sigma_k} \to \mathfrak{M}_{\mathbf{0}};$$
$$\hat{a}(b_1,\ldots,b_k) = ab_1\cdots b_k.$$

(ii) $\mathfrak{M}$ is λ-*closed* if for all $\varphi : \mathfrak{M}_{\vec{\sigma}} \to \mathfrak{M}_{\mathbf{0}}$

$$\varphi \text{ is } \lambda\mathfrak{M}\text{-definable } \Rightarrow\ \varphi = \hat{a} \text{ for some } a \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}.$$

It will turn out that if $\mathfrak{M}$ is λ-closed then $\mathfrak{P}_{\mathfrak{M}}$ is an extension of $\mathfrak{M}$, i.e., the 'first-order part' of $\mathfrak{P}_{\mathfrak{M}}$ is isomorphic to $\mathfrak{M}$.

We first need a technical result. Recall that $\to_{\mathfrak{M}}$ has been developed in stages.

We will show that for any $a \in \mathfrak{M}_\sigma$, the oracle $\boldsymbol{a}$ can be replaced by a term A representing a, in any computation yielding a numeral. We consider $\lambda\mathfrak{M}$-terms with *marked standard representations*, i.e. marked oracles (cf. Section 2.2) and marked numerals, in order to monitor these during computations. (The markings do not affect the reduction relations.) If M, A are terms, then $M[\underline{\underline{a}} := A]$ denotes M with all marked occurrences of $\underline{a}$ replaced by A.

3.4.6. PROPOSITION. *Let $\sigma \in \mathbb{T}_1$ and $a \in \mathfrak{M}_\sigma$. Let $A \in \Lambda\mathfrak{M}$ such that $A \rhd_{\beta\mathfrak{M}} a$. Then for all ζ one has*

(i) $\quad M \rhd_{\beta(\zeta)} b \ \Rightarrow\ M[\underline{\underline{a}} := A] \rhd_{\beta\mathfrak{M}} b.$

(ii) $\quad M \twoheadrightarrow_{\beta\zeta} N \ \Rightarrow\ M[\underline{\underline{a}} := A] =_{\beta\mathfrak{M}} N[\underline{\underline{a}} := A].$

PROOF. We prove (i), (ii) simultaneously by induction on the height of σ. Let M^* denote $M[\underline{\underline{a}} := A]$.

Basis ($\sigma \equiv \mathbf{0}$). Then (i), (ii) are trivial since

$$A \rhd_{\beta\mathfrak{M}} n \ \Leftrightarrow\ A =_{\beta\mathfrak{M}} \ulcorner n \urcorner.$$

Induction step. Let $\sigma \in \mathbb{T}_1$, and suppose the statement holds for all types of smaller height and for every ζ (we refer to this as IH_1). We prove the statement for σ by transfinite induction on ζ. Suppose the result holds for all $\vartheta < \zeta$ (this is IH_2).

As to (i), suppose $M \rhd_{\beta(\zeta)} b$. Let $\vec{c}$ be objects of appropriate types. Then $M\vec{c} \twoheadrightarrow_{\beta\vartheta} \ulcorner b\vec{c}\urcorner$ for some $\vartheta < \zeta$. Hence by IH_2(ii) one has

$$M^*\vec{c} =_{\beta\mathfrak{M}} \ulcorner b\vec{c}\urcorner.$$

We conclude that $M^* \rhd_{\beta\mathfrak{M}} b$.

As to (ii), we proceed by induction on the generation of $\twoheadrightarrow_{\beta\zeta}$. We only treat the prime cases. The compatibility and transitivity rules are easily verified.

- $M \to_{\beta\zeta} N$ is
$$(\lambda x.P)Q \to P[x := Q].$$
Then by substitutivity of $\to_{\beta\mathfrak{M}}$ one has $(\lambda x.P^*)Q^* \to_{\beta\mathfrak{M}} P^*[x := Q^*]$ and we are done.
- $M \to_{\beta\zeta} N$ is
$$\boldsymbol{a}'\vec{B} \to \ulcorner a'\vec{b}\urcorner \quad \text{since } \vec{B} \rhd_{\beta(\zeta)} \vec{b}.$$
(This includes $\boldsymbol{a}' \equiv \boldsymbol{a}$, i.e. an unmarked occurrence of $\boldsymbol{a}$.) Then $\vec{B}^* \rhd_{\beta(\zeta)} \vec{b}$ by (i), so $\boldsymbol{a}'\vec{B}^* \to_{\beta\mathfrak{M}} \ulcorner a'\vec{b}\urcorner$.
- $M \to_{\beta\zeta} N$ is
$$\underline{\boldsymbol{a}}\vec{B} \to \ulcorner a\vec{b}\urcorner \quad \text{since } \vec{B} \rhd_{\beta(\zeta)} \vec{b}.$$
Note that $A\vec{b} \twoheadrightarrow_{\beta\mathfrak{M}} \ulcorner a\vec{b}\urcorner$ because $A \rhd_{\beta\mathfrak{M}} a$. Moreover $\vec{B}^* \rhd_{\beta\mathfrak{M}} \vec{b}$ by (i). Therefore
$$A\vec{B}^* =_{\beta\mathfrak{M}} \ulcorner a\vec{b}\urcorner$$
by repeated application of IH_1(ii), marking the appropriate $b_{\underline{i}}$. □

Below we will often replace $=_{\beta\mathfrak{M}}$ by $=$ and $\rhd_{\beta\mathfrak{M}}$ by $\rhd$ if there is no danger of confusion.

3.4.7. COROLLARY. *Let* $a \in \mathfrak{M}_{\sigma\to\mathbf{0}}$.

(i) *Let* $b \in \mathfrak{M}_{\sigma_1}$. *Then*
$$A \rhd a,\ B \rhd b \ \Rightarrow\ AB \rhd ab.$$

(ii) *Let* $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$. *Then*
$$A \rhd a,\ \vec{B} \rhd \vec{b} \ \Rightarrow\ A\vec{B} = \ulcorner a\vec{b}\urcorner.$$

PROOF. (i) Suppose $A \rhd a$. Then for all $b, \vec{c}$ of the appropriate types
$$A\underline{b}\vec{c} = \ulcorner ab\vec{c}\urcorner,$$
so $A\underline{b} \rhd ab$. Hence $AB \rhd ab$ by Proposition 3.4.6 (i).

(ii) By repeated application of (i). □

3.4.8. CLASSIFICATION THEOREM. *Let $\mathfrak{M}$ be λ-closed. Then for all $\sigma \in \mathbb{T}_1$ one has the following in $\mathfrak{P}_{\mathfrak{M}}$, for each appropriate A and $a, a' \in \mathfrak{M}_\sigma$.*

$(\mathrm{i})_\sigma$ $\quad \langle\!\langle \underline{a} \rangle\!\rangle \in \mathrm{dom}[\sigma]$,

$(\mathrm{ii})_\sigma$ $\quad \langle\!\langle A \rangle\!\rangle \in \mathrm{dom}[\sigma] \Rightarrow A \rhd a$ *for some* a,

$(\mathrm{iii})_\sigma$ $\quad \langle\!\langle A \rangle\!\rangle\ [\sigma]\ \langle\!\langle \underline{a} \rangle\!\rangle \Leftrightarrow A \rhd a$,

$(\mathrm{iv})_\sigma$ $\quad \langle\!\langle \underline{a} \rangle\!\rangle\ [\sigma]\ \langle\!\langle \underline{a'} \rangle\!\rangle \Rightarrow a = a'$.

PROOF. By simultaneous induction on the height of σ.

Basis ($\sigma \equiv \mathbf{0}$). The properties $(\mathrm{i})_\mathbf{0}$ and $(\mathrm{ii})_\mathbf{0}$ hold by definition of $\mathcal{T}$ and $\rhd$ respectively. As to $(\mathrm{iii})_\mathbf{0}$, the ($\Rightarrow$) part holds by construction; for ($\Leftarrow$) use closedness of $\langle\!\langle A \rangle\!\rangle$ under $\beta\mathfrak{M}$-conversion. Moreover $(\mathrm{iv})_\mathbf{0}$ holds by the Church-Rosser theorem for $\beta\mathfrak{M}$-reduction.

Induction step ($\sigma \equiv \sigma_1 \to \cdots \to \sigma_k \to \mathbf{0}$). Note that $\underline{a} \equiv \boldsymbol{a}$ in this case.

(i) Let $a \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}$. Then for all $\vec{B} \rhd \vec{b}$

$$\underline{a}\vec{B} \equiv \boldsymbol{a}\vec{B} = \underline{a}\vec{b}.$$

Suppose $\langle\!\langle \vec{B} \rangle\!\rangle\ [\vec{\sigma}]\ \langle\!\langle \vec{B'} \rangle\!\rangle$. Then $\vec{B} \rhd \vec{b}$ for some $\vec{b}$, by $(\mathrm{ii})_{\vec{\sigma}}$, so $\langle\!\langle \vec{B} \rangle\!\rangle\ [\vec{\sigma}]\ \langle\!\langle \underline{\vec{b}} \rangle\!\rangle$ by $(\mathrm{iii})_{\vec{\sigma}}$. Similarly one has $\langle\!\langle \vec{B'} \rangle\!\rangle\ [\vec{\sigma}]\ \langle\!\langle \underline{\vec{b'}} \rangle\!\rangle$ for some $\vec{b'}$. But then $\vec{b} = \vec{b'}$ by $(\mathrm{iv})_{\vec{\sigma}}$. Hence

$$\langle\!\langle \boldsymbol{a} \rangle\!\rangle \cdot \langle\!\langle \vec{B} \rangle\!\rangle = \langle\!\langle \underline{a}\underline{\vec{b}} \rangle\!\rangle = \langle\!\langle \boldsymbol{a} \rangle\!\rangle \cdot \langle\!\langle \vec{B'} \rangle\!\rangle$$

and we are done.

(ii) Suppose $A \in \mathrm{dom}[\vec{\sigma}\to\mathbf{0}]$. Then for all $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$ one has by $(\mathrm{i})_{\vec{\sigma}}$

$$\langle\!\langle A \rangle\!\rangle \cdot \langle\!\langle \underline{\vec{b}} \rangle\!\rangle \in \mathrm{dom}[\mathbf{0}].$$

Hence for some $\varphi : \mathfrak{M}_{\vec{\sigma}} \to \mathfrak{M}_\mathbf{0}$ one has

$$A\underline{\vec{b}} = \underline{\varphi(\vec{b})}.$$

Using λ-closedness, determine $a \in \mathfrak{M}_{\vec{\sigma}\to\mathbf{0}}$ such that $\hat{a} = \varphi$. Then $A \rhd a$.

(iii) ($\Rightarrow$) Suppose $\langle\!\langle A \rangle\!\rangle\ [\sigma]\ \langle\!\langle \boldsymbol{a} \rangle\!\rangle$. Let $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$. By $(\mathrm{i})_{\vec{\sigma}}$ one has $\langle\!\langle \underline{\vec{b}} \rangle\!\rangle\ [\sigma]\ \langle\!\langle \underline{\vec{b}} \rangle\!\rangle$, so $\underline{\vec{b}} \rhd \vec{b}$ by $(\mathrm{iii})_{\vec{\sigma}}$. Now

$$\begin{aligned} \langle\!\langle A \rangle\!\rangle \cdot \langle\!\langle \underline{\vec{b}} \rangle\!\rangle &= \langle\!\langle \boldsymbol{a} \rangle\!\rangle \cdot \langle\!\langle \underline{\vec{b}} \rangle\!\rangle, \quad \text{by definition of } [\sigma] \\ &= \langle\!\langle \underline{a}\underline{\vec{b}} \rangle\!\rangle, \end{aligned}$$

so $A\underline{\vec{b}} = \underline{a}\underline{\vec{b}}$ by $(\mathrm{iv})_\mathbf{0}$. Therefore $A \rhd a$.

($\Leftarrow$) Suppose $A \rhd a$. Then for all $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$

$$A\underline{\vec{b}} = a\vec{b}.$$

By Corollary 3 4 7 (ii) one has for each $\vec{B} \rhd \vec{b}$

$$A\vec{B} = \underline{a\vec{b}}$$

Now suppose $\langle\!\langle \vec{B} \rangle\!\rangle\ [\vec{\sigma}]\ \langle\!\langle \vec{B'} \rangle\!\rangle$ Then it follows by $(\text{ii})_{\vec{\sigma}}$, $(\text{iii})_{\vec{\sigma}}$ and $(\text{iv})_{\vec{\sigma}}$ (cf the proof of (i)) that $\vec{B}, \vec{B'} \rhd \vec{b}$ for some $\vec{b}$ Therefore

$$\begin{aligned} \langle\!\langle A \rangle\!\rangle\ \langle\!\langle \vec{B} \rangle\!\rangle &= \langle\!\langle \underline{a\vec{b}} \rangle\!\rangle \\ &= \langle\!\langle \boldsymbol{a} \rangle\!\rangle\ \langle\!\langle \vec{B'} \rangle\!\rangle, \end{aligned}$$

so $\langle\!\langle A \rangle\!\rangle\ [\sigma]\ \langle\!\langle \boldsymbol{a} \rangle\!\rangle$

(iv) Suppose $\langle\!\langle \boldsymbol{a} \rangle\!\rangle\ [\sigma]\ \langle\!\langle \boldsymbol{a'} \rangle\!\rangle$ Then by $(\text{i})_{\vec{\sigma}}$ one has for all $\vec{b} \in \mathfrak{M}_{\vec{\sigma}}$

$$\boldsymbol{a}\vec{b} = \boldsymbol{a'}\vec{b}$$

so $a = a'$ by Church-Rosser and extensionality □

We will now show that each λ-closed $\mathfrak{M}$ is embeddable in $\mathfrak{P}_{\mathfrak{M}}$ (in the obvious sense) The first-order fragment of $\mathfrak{P}_{\mathfrak{M}}$ even corresponds exactly to $\mathfrak{M}$

3 4 9 Isomorphism Theorem *Let $\mathfrak{M}$ be λ-closed Then $\mathfrak{M} \cong \mathfrak{P}^1_{\mathfrak{M}}$*

Proof Define for each $\sigma \in \mathbb{T}_1$

$$f_\sigma \quad \mathfrak{M}_\sigma \to [\![\sigma]\!]$$

by

$$f_\sigma(a) = \langle\!\langle a \rangle\!\rangle \pmod{[\sigma]}$$

By the Classification Theorem this map is bijective (injectivity follows from $(\text{iv})_\sigma$ and surjectivity from $(\text{ii})_\sigma$) Moreover for all appropriate a, b

$$f_{\sigma\to\tau}(a)\ {}_{\sigma\,\tau}\ f_\sigma(b) = f_\tau(ab)$$

by (iii) of the Classification Theorem, using Corollary 3 4 7 (i) □

3.5. Interpretations in Extended Type Structures

In this section we consider interpretations in the per structures $\mathfrak{P}_{\mathfrak{M}}$ First, untyped terms can be interpreted in this structure

3 5 1 Definition Let $M \in \Lambda$ and let $\rho \quad V \to \mathcal{L}_{\mathfrak{M}}$ be a valuation, say $\mathrm{FV}(M) = \{x_1, \quad , x_n\}$ and $\rho(x_i) = \langle\!\langle P_i \rangle\!\rangle$ Then define

$$(\![M]\!)_\rho = \langle\!\langle M[\vec{x} := \vec{P}] \rangle\!\rangle$$

3.5.2. LEMMA. $(\![\ ]\!)$ *is a* λ*-interpretation in* $\mathfrak{P}_{\mathfrak{M}}$.

PROOF. Straightforward. □

We will use the denotation $\mathfrak{P}_{\mathfrak{M}}$ also for the per structure on $\mathfrak{M}$ extended with the standard interpretation above. Then we have that

$$\mathfrak{P}_{\mathfrak{M}} = \langle \mathcal{L}_{\mathfrak{M}},\ \cdot, \mathcal{T},\ (\![\ ]\!) \rangle$$

is a per structure for $\boldsymbol{\lambda 2}$.

3.5.3. THEOREM. $\mathfrak{P}_{\mathfrak{M}}$ *is a model of* $\boldsymbol{\lambda 2}$.

PROOF. Immmediate by Corollary 3.3.21. □

We will extend $\mathfrak{P}_{\mathfrak{M}}$ to per structures for other second-order systems $\boldsymbol{\lambda}\Box$ by extending the interpretation $(\![\ ]\!)$ according to a given constant valuation $\mathcal{V}$. The procedure is as follows.

3.5.4. DEFINITION. Let C be some set of constants, and $\mathcal{V} : C \to \mathcal{L}_{\mathfrak{M}}$ an interpretation of these, such that $(\mathcal{T}, \mathcal{V})$ is a constant valuation pair. Let $M \in \Lambda(C)$; say $c_1, \ldots, c_k$ are the constants appearing in M, and $\mathcal{V}(c_i) = \langle\!\langle Q_i \rangle\!\rangle$. Then $M^{\mathcal{V}}$ is obtained from M by replacing c_i by Q_i (for each i). Then define

$$(\![M]\!)^{\mathcal{V}}_{\rho} = \langle\!\langle M^{\mathcal{V}}[\vec{x} := \vec{P}] \rangle\!\rangle$$

where x_i, P_i are as in Definition 3.5.1. Now set

$$\mathfrak{P}^{\mathcal{V}}_{\mathfrak{M}} = \langle \mathcal{L}_{\mathfrak{M}},\ \cdot,\ \mathcal{T},\ \mathcal{V},\ (\![\ ]\!)^{\mathcal{V}} \rangle.$$

3.6. A Full Per Model

This section presents an application of the construction described in Section 3.4.

First of all, we can extend the $\boldsymbol{\lambda}\mathbf{T}$-model $\mathfrak{M}(\mathbb{N})$ to a model of $\boldsymbol{\lambda 2}\mathbf{T}$. Of course, $\mathfrak{M}(\mathbb{N})$ is extensional and λ-closed. Now we focus on

$$\mathfrak{P}_{\mathfrak{M}(\mathbb{N})} = \langle \mathcal{L}_{\mathfrak{M}(\mathbb{N})},\ \cdot,\ \mathcal{T} \rangle.$$

As was pointed out in Definition 3.5.4, we additionally have to specify $\mathcal{V}$.

Remember the implementation of conditionals and pairing in untyped λ-calculus.

3.6.1. DEFINITION. (i) Define

$$\begin{aligned} \textbf{true} &\equiv \lambda xy.x, \\ \textbf{false} &\equiv \lambda xy.y. \end{aligned}$$

Then $\textbf{true}\,MN = M$ and $\textbf{false}\,MN = N$, so if B is either equal to **true** or **false** the conditional if B then P else Q can be expressed as BPQ.

(ii) For $M, N \in \Lambda\mathfrak{M}$ the *ordered pair* $[M, N]$ is defined by setting

$$[M, N] \equiv \lambda z.zMN.$$

Note that

$$\begin{aligned} [M, N]\,\mathbf{true} &=_\beta M, \\ [M, N]\,\mathbf{false} &=_\beta N. \end{aligned}$$

We need some operations on the Church numerals.

3.6.2. DEFINITION. The terms $\mathbf{S}^+$, $\mathbf{P}^-$, and $\mathbf{Z}$ are defined as follows.

$$\begin{aligned} \mathbf{S}^+ &\equiv \lambda nfx.f(nfx), \\ \mathbf{P}^- &\equiv \lambda x.x(\lambda p.[\mathbf{S}^+(p\,\mathbf{true}), p\,\mathbf{true}])[\ulcorner 0 \urcorner, \ulcorner 0 \urcorner]\mathbf{false}, \\ \mathbf{Z} &\equiv \lambda x.x(\mathbf{K}\,\mathbf{false})\mathbf{true}. \end{aligned}$$

3.6.3. LEMMA. *For all* $n \in \mathbb{N}$

$$\begin{aligned} \mathbf{S}^+\ulcorner n \urcorner &= \ulcorner n+1 \urcorner, \\ \mathbf{P}^-\ulcorner n+1 \urcorner &= \ulcorner n \urcorner, \\ \mathbf{Z}\ulcorner 0 \urcorner &= \mathbf{true}, \\ \mathbf{Z}\ulcorner n+1 \urcorner &= \mathbf{false}. \end{aligned}$$

PROOF. Easy. □

The constants $\boldsymbol{R}_\sigma$ will be interpreted using a single lambda-calculus primitive recursor. This term serves as a uniform recursor: the oracles corresponding to recursors in $\mathfrak{M}(\mathbb{N})$ only have the proper behaviour on elements that are extensionally equal to $\mathfrak{M}(\mathbb{N})$-elements of the right type. For example, $\boldsymbol{R}_{\mathbf{PolyNat}}$ (see p. 33) cannot be interpreted by an oracle.

3.6.4. DEFINITION. Using the fixed point combinator $\mathbf{Y}$, define

$$\mathbf{Rec} \equiv \mathbf{Y}(\lambda rmnx.\mathsf{if}\ \mathbf{Z}\,x\ \mathsf{then}\ m\ \mathsf{else}\ n(\mathbf{P}^-x)(rmn(\mathbf{P}^-x))).$$

Then for all $M, N \in \Lambda\mathfrak{M}$ and $n \in \mathbb{N}$ one has

$$\begin{aligned} \mathbf{Rec}MN\ulcorner 0 \urcorner &= M, \\ \mathbf{Rec}MN\ulcorner n+1 \urcorner &= N\ulcorner n \urcorner(\mathbf{Rec}MN\ulcorner n \urcorner). \end{aligned}$$

3.6.5. DEFINITION. (i) The interpretation $\mathcal{V}$ of $\boldsymbol{\lambda 2}\mathbf{T}$-constants is as follows.

$$\begin{aligned} \mathcal{V}(\boldsymbol{0}) &= \langle\!\langle \ulcorner 0 \urcorner \rangle\!\rangle, \\ \mathcal{V}(\boldsymbol{S}) &= \langle\!\langle \mathbf{S}^+ \rangle\!\rangle, \\ \mathcal{V}(\boldsymbol{R}_\sigma) &= \langle\!\langle \mathbf{Rec} \rangle\!\rangle. \end{aligned}$$

(ii) The resulting structure is denoted by $\mathfrak{P}(\mathbb{N})$. So

$$\begin{aligned}\mathfrak{P}(\mathbb{N}) &= \mathfrak{P}^{\mathcal{V}}_{\mathfrak{M}(\mathbb{N})} \\ &= \langle \mathcal{L}_{\mathfrak{M}(\mathbb{N})}, \cdot, \mathcal{T}, \mathcal{V}, (\![\,]\!)^{\mathcal{V}} \rangle.\end{aligned}$$

3.6.6. LEMMA. *$(\mathcal{T}, \mathcal{V})$ is a constant valuation pair.*

PROOF. Let ξ be a type valuation in $\mathrm{PER}(\mathcal{L}_{\mathfrak{M}(\mathbb{N})})$. Obviously $\mathcal{V}(\boldsymbol{0}) \in \mathrm{dom}[\mathbf{0}]_\xi$. Moreover note that

$$\mathrm{dom}[\mathbf{0}\to\mathbf{0}]_\xi = \{\langle\!\langle F \rangle\!\rangle \mid F \in \Lambda^\circ\mathfrak{M}, F \rhd_{\beta\mathfrak{M}} f \text{ for some } f : \mathbb{N} \to \mathbb{N}\}.$$

Therefore $\mathcal{V}(\boldsymbol{S}) \in \mathrm{dom}[\mathbf{0}\to\mathbf{0}]_\xi$. Regarding $\boldsymbol{R}_\sigma$, suppose $\langle\!\langle M \rangle\!\rangle\ [\sigma]_\xi\ \langle\!\langle M' \rangle\!\rangle$, and $\langle\!\langle N \rangle\!\rangle\ [\mathbf{0}\to\sigma\to\sigma]_\xi\ \langle\!\langle N' \rangle\!\rangle$, and prove by induction on n that

$$\langle\!\langle \mathbf{Rec} \rangle\!\rangle \langle\!\langle M \rangle\!\rangle \langle\!\langle N \rangle\!\rangle \langle\!\langle \ulcorner n \urcorner \rangle\!\rangle\ [\sigma]_\xi\ \langle\!\langle \mathbf{Rec} \rangle\!\rangle \langle\!\langle M' \rangle\!\rangle \langle\!\langle N' \rangle\!\rangle \langle\!\langle \ulcorner n \urcorner \rangle\!\rangle.$$

It follows that $\mathcal{V}(\boldsymbol{R}_\sigma) \in \mathrm{dom}[\sigma\to(\mathbf{0}\to\sigma\to\sigma)\to\mathbf{0}\to\sigma]_\xi$. □

So we can conclude that $\mathfrak{P}(\mathbb{N})$ is a per structure for $\boldsymbol{\lambda 2T}$.

3.6.7. THEOREM. *$\mathfrak{P}(\mathbb{N})$ is a model of $\boldsymbol{\lambda 2T}$.*

PROOF. In view of Proposition 3.3.20 we only have to check the validity of (CR). This is verified by translating the equations following Definition 3.6.4 to $\mathcal{L}_{\mathfrak{M}}$. □

In Breazu-Tannen and Meyer (1987), the authors investigate plain second order extensions $(\boldsymbol{\lambda}\Box)^2$ of first-order theories $\boldsymbol{\lambda}\Box$ (being $\boldsymbol{\lambda}^\tau$-theories over a many-sorted algebra). They show that these extensions are conservative with respect to provable equality, using (a refinement of) the property that every closed $(\boldsymbol{\lambda}\Box)^2$-term of simple type has a normal form in $\boldsymbol{\lambda}\Box$. As a consequence of this property, there is no increase in computational strength in the simple types of the polymorphic extension. In Section 5 of their paper, the authors mention this aspect of their construction, which they consider unfortunate.

As a second result, Breazu-Tannen and Meyer essentially show that every extensional model of a theory $\boldsymbol{\lambda}\Box$ can be extended to a model of $(\boldsymbol{\lambda}\Box)^2$. At first sight, the construction of e.g. our model $\mathfrak{P}^{\mathcal{V}}_{\mathfrak{M}(\mathbb{N})}$ for $\boldsymbol{\lambda 2T}$ seems to overlap with their approach. The situation here, however, is essentially different.

Breazu-Tannen and Meyer again use the normalization property mentioned above. This does not apply to our case: we have already seen that $\boldsymbol{\lambda 2T}$ is not the plain polymorphic extension of $\boldsymbol{\lambda T}$; the absence of increase of computational power contrasts Corollary 3.1.11.

Moreover, the construction in Breazu-Tannen and Meyer (1987) is essentially *typed*, whereas our first-order model has to be encoded into an *untyped* applicative

structure The fact that a per construction over an untyped applicative structure is powerful enough is interesting in itself

The primary motivation of our construction was to build a certain counter model, to be used in Chapter 4, and not to improve on Breazu Tannen and Meyer (1987) However, for our model of $\boldsymbol{\lambda 2T}$ we use a construction which may be seen as an improvement of the construction of Breazu Tannen and Meyer (1987) in that it does not suffer from the unfortunate aspect mentioned above Of course, there is a price to be paid First, the first-order model has to satisfy strong computational closure conditions Second, our construction is much more involved than that in Breazu-Tannen and Meyer (1987)

The fact that $\mathfrak{M}(\mathbb{N})$ can be extended to a model of $\boldsymbol{\lambda 2T}$ does *not* contradict Reynolds (1984) 'In this paper, we will prove that the standard set-theoretic model of the ordinary typed lambda calculus cannot be extended to model this [polymorphic] language extension ' In fact the latter quote does not capture the main result of Reynolds (1984), which states that the type constructor $\rightarrow$ in the polymorphic lambda calculus cannot be interpreted as the set theoretic function space constructor Our results show that $\rightarrow$ can be interpreted in such a way that its restriction to simple types corresponds (up to isomorphism) to the function space constructor

3.7. Kleene Recursion

Using the S-m-n Theorem and the Recursion Theorem from classical recursion theory, one can show that the type structure HEO is λ closed The same can be done for the structure ECF of extensional continuous functionals, by applying analogues of the S-m-n Theorem and the Recursion Theorem We will not go into this here

The λ-closure property of ω-structures can be reformulated into more familiar terms by considering the notion of recursiveness in higher types given by Kleene (1959b), for partial functions

$$\varphi \quad \mathfrak{M}_{\sigma_1} \times \quad \times \mathfrak{M}_{\sigma_k} \rightharpoonup \mathfrak{M}_0$$

We recapitulate Kleene's definition below For an easy presentation we restrict the σ_i (following Kleene) to the pure types $\mathbf{0}, \mathbf{1}, \mathbf{2},$ The extension to arbitrary simple types is straightforward but requires tedious administration

We borrow some notation and terminology from Kechris and Moschovakis (1977)

3 7 1 DEFINITION (i) Each sequence $\vec{a}$ of arguments (of pure types) is assumed to be in *simplified form*, i e objects of lower type appear before objects of higher type So only the order of objects of the same type matters

(ii) If b is an object of pure type j, then $(b, \vec{a})$ denotes the simplied sequence consisting of the mentioned objects having b as first type-j object

(iii) If $\vec{a}$ is a sequence of objects of pure types, then arity(a) is the (encoded) sequence

$$\langle n_0, n_1, \ldots, n_r \rangle$$

where r is the highest pure type occuring in $\vec{a}$, and n_j is the number of type-j objects in $\vec{a}$.

Kleene's definition of recursiveness is index oriented: the schemata (S1)-(S9) describe indices $e \in \mathbb{N}$ for partial recursive functions on pure types, and define the relation

$$\{e\}(\vec{a}) \simeq n$$

inductively. The index dependency manifests itself in the schema (S9).

3.7.2. DEFINITION. For each $e \in \mathbb{N}$ and sequence $\vec{a}$ of pure types, the result of applying $\{e\}$ to $\vec{a}$ is defined by the following schemata.

(S1)	$\{e\}(x, \vec{a})$	$\simeq$	$x+1$	if $e = \langle 1, \text{arity}(x, \vec{a}) \rangle$ (*successor*)
(S2)	$\{e\}(\vec{a})$	$\simeq$	q	if $e = \langle 2, \text{arity}(\vec{a}), q \rangle$ (*constant functions*)
(S3)	$\{e\}(x, \vec{a})$	$\simeq$	x	if $e = \langle 3, \text{arity}(x, \vec{a}) \rangle$ (*projection*)
(S4)	$\{e\}(\vec{a})$	$\simeq$	$\{e_2\}(\{e_1\}(\vec{a}), \vec{a})$	if $e = \langle 4, \text{arity}(\vec{a}), e_1, e_2 \rangle$ (*composition*)
(S5)	$\{e\}(0, \vec{a})$	$\simeq$	$\{e_1\}(\vec{a})$	if $e = \langle 5, \text{arity}(x, \vec{a}), e_1, e_2 \rangle$
	$\{e\}(x+1, \vec{a})$	$\simeq$	$\{e_2\}(x, \{e\}(x, \vec{a}), \vec{a})$	(*primitive recursion*)
(S6)	$\{e\}(\vec{a})$	$\simeq$	$\{e_1\}([\vec{a}]^j_k)$	if $e = \langle 6, \text{arity}(\vec{a}), e_1, j, k \rangle$ (*shift*)

where $\vec{a}$ contains at least $k+1$ type-j objects, and $[\vec{a}]^j_k$ results from $\vec{a}$ by moving the first type-j argument k places to the right.

(S7)	$\{e\}(x, f^{\mathbf{1}}, \vec{a})$	$\simeq$	$f \cdot x$	if $e = \langle 7, \text{arity}(x, f, \vec{a}) \rangle$ (*type-0 application*)
(S8)	$\{e\}(b^{j+2}, \vec{a})$	$\simeq$	$b \cdot (\lambda c^j.\{e_1\}(b, c, \vec{a}))$	if $e = \langle 8, \text{arity}(b, \vec{a}), e_1 \rangle$ (*higher type application*)

where the b-application is to be interpreted as $b \cdot d$ if

$$\widehat{d} = \boldsymbol{\lambda} c^j.\{e_1\}(b, c, \vec{a})$$

provided such a d exists, and undefined otherwise.

(S9)	$\{e\}(x, \vec{a}, \vec{b})$	$\simeq$	$\{x\}(\vec{a})$	if $e = \langle 9, \text{arity}(x, \vec{a}, \vec{b}), \text{arity}(\vec{a}) \rangle$ (*reflection*)

Moreover $\{e\}(\vec{a})$ is undefined if none of the above clauses applies.

3.7.3. DEFINITION. Let $\varphi : \mathfrak{M}_{\sigma_1} \times \cdots \times \mathfrak{M}_{\sigma_k} \rightharpoonup \mathfrak{M}_{\mathbf{0}}$.

(i) φ is *partial Kleene recursive* if for some e one has

$$\forall \vec{a} \quad \varphi(\vec{a}) \simeq \{e\}(\vec{a}).$$

Such a φ is called *Kleene recursive* if φ is total. The notion of relative recursiveness ('*recursive in* $\vec{b}$') is defined in the obvious way. Moreover φ is said to be *recursive in* $\mathfrak{M}$ if it is recursive in some $\vec{b} \in \mathfrak{M}$.

(ii) $\mathfrak{M}$ is *Kleene closed* if for any φ

$$\varphi \text{ is Kleene recursive in } \mathfrak{M} \Rightarrow \varphi = \hat{a} \text{ for some } a \in \mathfrak{M}.$$

3.7.4. THEOREM. *$\mathfrak{M}$ is λ-closed iff $\mathfrak{M}$ is Kleene closed.*

PROOF (Sketch). One can adapt the argument given by Kleene (1959b) to prove that φ is $\lambda\mathfrak{M}$-definable by some F with oracles from $\vec{b}$ iff φ is Kleene recursive in $\vec{b}$. □

Chapter 4

Bar Recursion versus Polymorphism

4.1. Introduction

Spector (1962) extended Godel's functional interpretation of arithmetic (Godel (1958)) to analysis This involved a new concept in higher order subrecursion theory called *bar recursion* Bar recursion is essentially a principle of definition for functionals of simple type through transfinite recursion over a well-founded tree of finite sequences of functionals Spector's results yielded, apart from a consistency proof for analysis, a characterization of the provably total recursive functions (resp the provable well-orderings) of analysis as those functions (resp well-orderings) that are definable by bar recursion

In his thesis, Girard (1972) extended Godel's results cited above to second and higher order intuitionistic arithmetic This involved a completely new system of typed lambda calculi, namely second and higher order (typed) lambda calculus The second order theory has been introduced as $\boldsymbol{\lambda 2T}$ in Section 3 1 Apart from consistency proofs for second and higher order arithmetic, Girard's results yielded characterizations of the provably total recursive functions of these theories in terms of definability in the corresponding typed lambda calculi

The metamathematical results from Spector (1962) and Girard (1972) imply that the class of bar recursive functions coincides with the class of functions definable in second order typed lambda calculus This phenomenon suggests a, possibly deep, relationship between bar recursion and second order lambda cal culus, and calls for an explanation As a special instance of this general problem one could consider the question of definability of *functionals* of higher type (as opposed to *functions*) in both systems

We describe a notion of definability of functionals based on specifications in the language of Godel's system $\boldsymbol{\lambda T}$ of higher order primitive recursive functionals, with a (semantical) notion of realizability of these specifications For type $\mathbf{1}$ this yields exactly the definability results of Spector (1962) and Girard (1972)

We present a specification $\Sigma = \Sigma(\Phi)$, with Φ a type **3** variable The computational interpretation of $\Sigma(\Phi)$ is that Φ is a certain minimization functional It will be shown that Σ has a realization in Spector's system λ**TB**, but is not realizable in Girard's system λ**2T** This shows that (with our notion of realizability) λ**TB** and λ**2T** have different classes of functionals (at least of type **3**) The case of type **2** is open It is also open whether there exists a type **3** functional which is definable in λ**2T** but not in λ**TB**, but we consider this unlikely There is evidence that λ**TB** is in some sense stronger than λ**2T** this latter system interprets a purely intuitionistic theory, whereas the former also interprets a weak form of the Law of the Excluded Middle (Double Negation Shift, or axiom F from Spector (1962))

The positive result for λ**TB** is proved by a purely syntactical argument it turns out that there is a bar recursive term that provably satisfies the specification Σ Surprisingly, Σ has a realization in the language of λ**T**, but only provably so in λ**TB** (or even in weaker calculi such as λ**T** extended with a fan functional) This yields some non-conservativity results

The proof of the negative result concerning λ**2T** is carried out by applying the model construction in Chapter 3 to obtain a *counter model*, i e a model of λ**2T** in which our specification Σ has no solution (i e $\exists\Phi\ \Sigma(\Phi)$ is false in the counter model) A key property of this model is the presence of a specific discontinuous functional of type **2** (with type **1** equipped with the Baire topology, and type **0** with the discrete topology), which seems to be absent in all usual models of λ**2T**

4.2. Syntax

In this chapter, we consider extensions of λ**T** with bar recursion (λ**TB**) and with polymorphism (λ**2T**) To the systems λ**T**, λ**TB** and λ**2T** one can add quantifier-free arithmetic, which will be expressed by the denotation $\lambda\Box_{\mathbf{A}}$ With $\longrightarrow$ expressing the subsystem relation we can depict the situation as follows

$$\begin{array}{lll} \lambda^{\tau} \longrightarrow \lambda\mathbf{T}_{(\mathbf{A})} \longrightarrow \lambda\mathbf{TB}_{(\mathbf{A})} & & \text{first order systems} \\ \downarrow \qquad\quad \downarrow & & \\ \lambda\mathbf{2} \longrightarrow \lambda\mathbf{2T}_{(\mathbf{A})} & & \text{second order systems} \end{array}$$

Bar Recursion

In this chapter we only consider bar recursion of the lowest possible type This bar recursion (of type **0**) is in fact recursion on trees of finite sequences (of natural numbers) See Bezem (1986) for more details on general bar recursion

4 2 1 DEFINITION (i) The collection of *finite sequences* of natural numbers is indicated by Seq We use the denotation $\langle n_0, \quad , n_{k-1}\rangle$ $(k \geq 0)$ for elements of

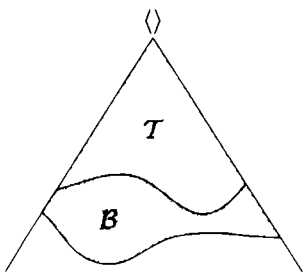
Seq; the *empty sequence* is denoted by $\langle\,\rangle$. Let $s, s', \ldots$ range over such sequences.

(ii) We presuppose a surjective, primitive recursive *encoding* of such sequences as natural numbers, with $*$ (*concatenation* operator), lh (*length* function) and $\preceq$ (*prefix* relation) primitive recursive.

(iii) If $s = \langle n_0, \ldots, n_{k-1}\rangle$ then $[s]$ is the function assigning n_i to $i < k$ and 0 to $i \geq k$. Note that $[s]$ is primitive recursive in s.

(iv) If $f \in \mathbb{N}^{\mathbb{N}}$ and $n \in \mathbb{N}$, then $\overline{f}(n)$ is the sequence $\langle f(0), \ldots, f(n-1)\rangle$.

For an informal explanation of bar recursion, consider the tree of sequence numbers, ordered by the prefix relation.



Now suppose $\mathcal{B} \subseteq \mathrm{Seq}$ is a *bar*, i.e.

$$\forall f \in \mathbb{N}^{\mathbb{N}}\ \exists n \in \mathbb{N}\ [f(n) \in \mathcal{B}].$$

Then we can specify a function F on the tree

$$\mathcal{T} = \{s \in \mathrm{Seq} \mid \forall s' \prec s\ [s' \notin \mathcal{B}]\}$$

by specifying F on the leaves of $\mathcal{T}$ (or even on all elements of $\mathcal{B}$), and defining F on each inner node s recursively in terms of all values $F(s * \langle x\rangle)$. In particular, F is defined in $\langle\,\rangle$.

Technically, F is said to be *bar recursive in G and H* if Gs gives the value of F at each leaf s, and the value in an inner node s is obtained by applying Hs to $\lambda x.F(s * \langle x\rangle)$ (all predecessors of s in $\mathcal{T}$).

In $\boldsymbol{\lambda}\mathbf{TB}$, a bar is given by a type-**2** functional Y, by setting

$$\mathcal{B}_Y = \{s \in \mathrm{Seq} \mid Y[s] < \mathrm{lh}(s)\}.$$

Of course each Y should be such that $\mathcal{B}_Y$ is a proper bar (i.e., the corresponding tree $\mathcal{T}$ is well-founded). This indicates that it is non-trivial to find a model of bar recursion.

Now we can formally introduce $\boldsymbol{\lambda}\mathbf{TB}$.

4 2 2 DEFINITION The system $\boldsymbol{\lambda TB}$ is the extension of $\boldsymbol{\lambda T}$ with the constant

$$\boldsymbol{B} \quad \text{of type} \quad \mathbf{2}\to\mathbf{1}\to(\mathbf{0}\to\mathbf{1}\to\mathbf{0})\to\mathbf{0}\to\mathbf{0}$$

The intended interpretation is that of *bar recursor* To the axioms and rules of $\boldsymbol{\lambda T}$ we add the following

$$\frac{Y[s] < \mathrm{lh}(s)}{\boldsymbol{B}YGHs = Gs} \qquad \frac{Y[s] \geq \mathrm{lh}(s)}{\boldsymbol{B}YGHs = Hs(\lambda x^{\mathbf{0}}\, \boldsymbol{B}YGH(s * \langle x\rangle))} \tag{CB}$$

The denotations $\mathrm{lh}(s)$, $[s]$ and $*$, $\langle x\rangle$ in the above rules should be taken as abbreviations of $\boldsymbol{\lambda T}$-terms representing the corresponding primitive recursive functions The premisses can be considered equational, using the (primitive recursive) characteristic functions of $<$ and $\geq$, as will be explained below

Quantifier-free Arithmetic

To increase the proof-theoretic usability of our theories we can extend these with rules for the successor function and for induction, and with rules for propositional logic This leads to systems arithmetical extensions $\boldsymbol{\lambda\Box_A}$

4 2 3 DEFINITION The additional rules for the structure of natural numbers are the following

$$\frac{\boldsymbol{S}P = \boldsymbol{S}Q}{P = Q}\,\mathrm{SE} \qquad \frac{\boldsymbol{S}P = \boldsymbol{0}}{\varphi}\,\bot\mathrm{R} \qquad \frac{\varphi(\boldsymbol{0}, y) \qquad \varphi(\boldsymbol{S}x, y)}{\varphi(P, Q)}\,\mathrm{IR} \tag{N}$$

The conclusions in the above rules are schematic in order to avoid a general instantiation rule It will be proved that full instantiation is derivable in the system (see Corollary 4 2 13)

Our induction rule is a little special, which we shall explain now Firstly, the second premiss (the induction step) is casted so that it can be added to purely equational systems This is possible because we allow derivations to depend on assumptions Some additional care in dealing with variables should be exercised Of course the induction rule can only be applied when no variable in P or Q occurs free in any assumption (other than the induction hypothesis $\varphi(x, Fy)$) on which the premisses depend

Secondly, a more usual form of the induction rule is obtained by putting $F \equiv \lambda y\, y$ so that $Fy = y$ Since φ may contain other variables than those explicitly shown, we can formulate this induction rule as usual

$$\frac{\varphi(\boldsymbol{0}) \qquad \overset{\varphi(x)}{\varphi(\boldsymbol{S}x)}}{\varphi(P)}\,\mathrm{IR}'$$

In the rule IR the induction hypothesis $\varphi(x, Fy)$ has as special feature the occurrence of the term F. Note that the rule is sound due to the base step $\varphi(\mathbf{0}, y)$ (and the fact that y does not occur free in any assumption on which the premiss depends). The reason for the special feature is that it allows us to prove conveniently in our (quantifier-free) theory some lemmas whose usual proofs are by double induction, which cannot be done in a quantifier-free system. Actually, the rule IR is a derived rule in the system with the at first sight weaker rule IR′. The tedious proof of this fact (essentially formalizing the semantic argument given in the proof of Theorem 4.3.1, cf. Troelstra (1973), 1.7.8–10) is avoided by postulating the stronger rule.

The ease in use of the quantifier-free arithmetic would be greatly improved by the addition of propositional logic. This should be done with care in order to allow the formulas to be translated into purely equational form. We first give a precise definition of the set $\mathrm{Form}(\boldsymbol{\lambda\Box_A})$ of (propositional) formulas.

4.2.4. DEFINITION. Let $\boldsymbol{\lambda\Box}$ be one of the theories $\boldsymbol{\lambda T}$, $\boldsymbol{\lambda TB}$, $\boldsymbol{\lambda 2T}$. The set $\mathrm{Form}_\sigma(\boldsymbol{\lambda\Box_A})$ of *formulas of type* σ is inductively defined as follows.

$$M, N \in \mathrm{Term}_\sigma(\boldsymbol{\lambda\Box}) \;\Rightarrow\; (M =_\sigma N) \in \mathrm{Form}_\sigma(\boldsymbol{\lambda\Box_A}),$$
$$\varphi, \psi \in \mathrm{Form}_{\mathbf{0}}(\boldsymbol{\lambda\Box_A}) \;\Rightarrow\; (\varphi \to \psi) \in \mathrm{Form}_{\mathbf{0}}(\boldsymbol{\lambda\Box_A}).$$

As before, $\varphi, \psi, \chi, \varphi'$ range over formulas. To economize on parentheses, outermost parentheses are omitted and we take $\to$ to associate to the right. It is important to stress that only type $\mathbf{0}$ equations may be combined into propositional formulas (for reasons that will become clear below). We single out the formula $\bar{0} = \bar{1}$ from $\mathrm{Form}_{\mathbf{0}}(\boldsymbol{\lambda\Box_A})$ and denote it by $\bot$.

4.2.5. DEFINITION. The rules for *propositional logic* read as follows.

$$\frac{\begin{array}{c}[\varphi]\\ \vdots\\ \psi\end{array}}{\varphi \to \psi}\to\mathrm{I} \qquad\qquad \frac{\varphi \to \psi \qquad \varphi}{\psi}\to\mathrm{E} \tag{P}$$

4.2.6. DEFINITION. The extension of $\boldsymbol{\lambda\Box}$ with *quantifier-free arithmetic* (notation $\boldsymbol{\lambda\Box_A}$) is obtained by adding the rules (N) and (P). The corresponding (extended) deduction relation is denoted by $\vdash_{\boldsymbol{\lambda\Box_A}}$. Note that the induction rule now also applies to propositional formulas.

The definitions of the theories $\boldsymbol{\lambda\Box}$ are summarized in the following table. The rules L1, L2, and CR have been introduced in Section 3.1.

$\boldsymbol{\lambda^\tau}$	L1					
$\boldsymbol{\lambda T_{(A)}}$	L1		CR		(N	P)
$\boldsymbol{\lambda TB_{(A)}}$	L1		CR	CB	(N	P)
$\boldsymbol{\lambda 2}$	L1	L2				
$\boldsymbol{\lambda 2T_{(A)}}$	L1	L2	CR		(N	P)

The following definitions and lemmas prepare for a translation from formulas of $\boldsymbol{\lambda\Box}_{\mathbf{A}}$ into equations while preserving provability.

4.2.7. DEFINITION. We introduce the following abbreviations for terms.

$$\begin{aligned}
\mathsf{C} &\equiv \lambda x^0 y^0 z^0.\boldsymbol{R}_0 z(\lambda n^0 m^0.y)x \quad \text{(conditional)},\\
\mathsf{P} &\equiv \boldsymbol{R}_0\bar{0}(\lambda x^0 y^0.x) \quad \text{(predecessor)},\\
\boxed{-} &\equiv \lambda x^0 y^0.\boldsymbol{R}_0 x(\lambda u^0 v^0.\mathsf{P}v)y \quad \text{(cut-off subtraction)},\\
\boxed{<} &\equiv \lambda x^0 y^0.\boldsymbol{R}_0\bar{0}(\lambda u^0 v^0.\bar{1})(y\boxed{-}x) \quad \text{(less than)},\\
\boxed{=} &\equiv \lambda x^0 y^0.\boldsymbol{R}_0(\boldsymbol{R}_0\bar{1}(\lambda u^0 v^0.\bar{0})(x\boxed{-}y))(\lambda u^0 v^0.0)(y\boxed{-}x) \quad \text{(equal)},\\
\boxed{\to} &\equiv \lambda x^0 y^0.\boldsymbol{R}_0 1(\lambda u^0 v^0.y)x \quad \text{(implication)}.
\end{aligned}$$

Note that primitive recursion of the lowest type suffices for these operations. Observe that we write $\boxed{-}$ as an infix operator (binding weaker than unary functions).

4.2.8. DEFINITION. We introduce the following abbreviations for formulas.

$$\begin{aligned}
M \neq N &\equiv (M = N) \to \bot,\\
M < N &\equiv \boxed{<}MN = \bar{1},\\
M > N &\equiv \boxed{<}NM = \bar{1}.
\end{aligned}$$

Furthermore, $\Gamma \vdash_{\boldsymbol{\lambda\Box}} \varphi \leftrightarrow \psi$ is shorthand for $\Gamma \vdash_{\boldsymbol{\lambda\Box}} \varphi \to \psi$ and $\Gamma \vdash_{\boldsymbol{\lambda\Box}} \psi \to \varphi$. We abbreviate $\vdash_{\boldsymbol{\lambda T}_{\mathbf{A}}}$ by $\vdash$.

4.2.9. LEMMA. *The following can be proved in* $\boldsymbol{\lambda T}_{\mathbf{A}}$.

(i) $\vdash x \neq \bar{0} \to x = \boldsymbol{S}(\mathsf{P}\,x)$
(ii) $\vdash (x \neq \bar{0} \to \bot) \to x = \bar{0}$
(iii) $\vdash x \neq \bar{0} \leftrightarrow x > \bar{0}$
(iv) $\vdash \bar{0}\boxed{-}x = x$
(v) $\vdash \boldsymbol{S}M\boxed{-}x = \bar{0} \to \bar{0} < x$
(vi) $\vdash \boldsymbol{S}x\boxed{-}\boldsymbol{S}y = x\boxed{-}y$
(vii) $\vdash (x \neq y \to \bot) \to x = y$
(viii) $\vdash x\boxed{-}y = \bar{0} \to y\boxed{-}x = \bar{0} \to x = y$
(ix) $\vdash x\boxed{-}x = \bar{0}$
(x) $\vdash x = y \leftrightarrow \boxed{=}xy = \bar{1}$
(xi) $\vdash x \neq y \leftrightarrow \boxed{=}xy = \bar{0}$
(xii) $\vdash \boxed{\to}(\boldsymbol{S}x)y = y$
(xiii) $\vdash \boxed{\to}\bar{0}y = \bar{1}$
(xiv) $\vdash \boxed{\to}xy = \bar{0} \to x \neq \bar{0}$
(xv) $\vdash \boxed{\to}xy = \bar{0} \to y = \bar{0}$
(xvi) $\vdash \boxed{\to}xy = \bar{1} \to y \neq \bar{1} \to x = \bar{0}$
(xvii) $y \neq \bar{0} \to y = \bar{1} \vdash \boxed{\to}xy \neq \bar{0} \to \boxed{\to}xy = \bar{1}$

PROOF. The proofs are more or less standard and can be found in the literature (cf. Troelstra (1973), 1.7.2–7, where this development of quantifier-free arithmetic is attributed to K. Schütte). We leave them as exercises to the reader, but not without providing a number of hints. Most proofs use the induction rule IR as the essential step. However, not all proofs use IR in the same way. We distinguish between the following cases. First consider the case in which the special feature of our induction rule is not used, so in fact only the weaker rule IR′ is applied. If no induction hypothesis is used at all, then we actually do only case distinction (cases $\varphi(\boldsymbol{0})$ and $\varphi(\boldsymbol{S}x)$) and will phrase this accordingly. If we do use the induction hypothesis $\varphi(x)$, then we phrase the case as 'by ordinary induction'. Next consider the case that the force of the induction rule with parameters is indeed used. Then we phrase the proof as 'by induction' and mention the induction hypothesis explicitly. Moreover we mention which variable is the induction variable and which previous parts of the lemma can be profitably used. Hints for the proofs: (i)–(v) by case distinction on x; (vi) by ordinary induction on y; (vii) by induction on y using the induction hypothesis ($\mathbf{P}x \neq y \to \bot) \to \mathbf{P}x = y$ and part (ii); (viii) by induction on x using the induction hypothesis $x\boxed{-}\mathbf{P}y = \bar{0} \to \mathbf{P}y\boxed{-}x = \bar{0} \to x = \mathbf{P}y$ and the parts (i), (iii) and (v); (ix) by ordinary induction on x using (vi); (x),(xi) by previous parts (not using the induction rule at all); (xii)–(xiv) trivial; (xv)–(xvii) by case distinction on x. □

4.2.10. DEFINITION. We define a mapping

$$\varphi \mapsto \boxed{\varphi} \quad . \quad \mathrm{Form}_0(\boldsymbol{\lambda}\Box_{\mathbf{A}}) \to \mathrm{Term}_0(\boldsymbol{\lambda}\Box_{\mathbf{A}})$$

by induction on the structure of φ as follows.

$$\begin{aligned} \boxed{M =_0 N} &\equiv \boxed{=}MN, \\ \boxed{\varphi \to \psi} &\equiv \boxed{\to}\boxed{\varphi}\boxed{\psi}. \end{aligned}$$

4.2.11. THEOREM. *For all* $\varphi \in \mathrm{Form}_0(\boldsymbol{\lambda}\Box_{\mathbf{A}})$ *we have*

(i) $\mathrm{FV}(\varphi) = \mathrm{FV}(\boxed{\varphi})$;

(ii) $\vdash_{\boldsymbol{\lambda}\mathbf{T}_{\mathbf{A}}} \boxed{\varphi} \neq \bar{0} \to \boxed{\varphi} = \bar{1}$.

(iii) $\vdash_{\boldsymbol{\lambda}\mathbf{T}_{\mathbf{A}}} \varphi \leftrightarrow \boxed{\varphi} = \bar{1}$.

PROOF. (i) is obvious. (ii) and (iii) are proved by induction on φ, using Lemma 4.2.9. □

4.2.12. COROLLARY. $\boldsymbol{\lambda}\Box_{\mathbf{A}}$ *enjoys the full force of classical propositional logic.*

PROOF. By Theorem 4.2.11 (iii) and Lemma 4.2.9 (vii). □

4.2.13. COROLLARY. *If the variable x does not occur free in Γ, then*

$$\Gamma \vdash_{\lambda\Box_{\mathbf{A}}} \varphi \;\Rightarrow\; \Gamma \vdash_{\lambda\Box_{\mathbf{A}}} \varphi[x := N].$$

PROOF. First assume φ is an equation, say $P = Q$. Since x does not occur in Γ it follows by the ξ-rule that $\lambda x.P = \lambda x.Q$. Now we calculate $P[x := N] = (\lambda x.P)N = (\lambda x.Q)N = Q[x := N]$, so $\varphi[x := N]$. If φ is not an equation, then apply Theorem 4.2.11 (iii) and the fact that $\underline{\varphi[x := N]} \equiv \underline{\varphi}[x := N]$. $\Box$

4.2.14. REMARK. Theorem 4.2.11 above cannot be generalized to formulas $M =_\sigma N$ with $\sigma \neq \mathbf{0}$. The reason is that the higher type equalities are undecidable, whereas equality of type $\mathbf{0}$ is decidable (even primitive recursive). As a consequence, adding propositional logic for equations of arbitrary type would violate the equational character of the theory.

4.2.15. REMARK. It is useful to remark that for all $\varphi \in \mathrm{Form}_0(\boldsymbol{\lambda}\Box_{\mathbf{A}})$ there exist $\boldsymbol{\lambda}\Box$-terms $\mu x{\leq}y.\ \varphi$, $\forall x{\leq}y.\ \varphi$, $\exists x{\leq}y.\ \varphi$ encoding, respectively, bounded minimization, bounded universal quantification and bounded existential quantification. More precisely, these terms satisfy, provably in $\boldsymbol{\lambda}\mathbf{T}_{\mathbf{A}}$, the following specifications.

$$\begin{aligned}
(\mu x{\leq}y.\ \varphi) < \boldsymbol{S}y &\;\rightarrow\; \varphi[x := (\mu x{\leq}y.\ \varphi)],\\
x < (\mu x{\leq}y.\ \varphi) &\;\rightarrow\; \varphi \rightarrow \bot,\\
(\exists x{\leq}y.\ \varphi) = \bar{1} &\;\rightarrow\; (\mu x{\leq}y.\ \varphi) < \boldsymbol{S}y,\\
(\exists x{\leq}y.\ \varphi) = \bar{0} &\;\rightarrow\; x \leq y \rightarrow \varphi \rightarrow \bot,\\
(\forall x{\leq}y.\ \varphi) = 1 &\;\rightarrow\; x \leq y \rightarrow \varphi,\\
(\forall x{\leq}y.\ \varphi) = \bar{0} &\;\rightarrow\; (\mu x{\leq}y.\ \varphi \rightarrow \bot) < \boldsymbol{S}y.
\end{aligned}$$

Our formulation of $\boldsymbol{\lambda}\mathbf{TB}_{\mathbf{A}}$ corresponds to that of Howard (1968). In order to transfer our results to purely equational systems (i.e., without (P)) one should work with the encoded propositional formulas $\underline{\varphi}$, cf. Spector (1962). This requires an additional conservativity result for the propositional extension, which goes back to Goodstein (1941).

4.3. Semantics

We do not need models for $\boldsymbol{\lambda}\mathbf{TB}$ in this chapter, since the (positive) results concerning that system can be proved syntactically. One usually considers the type structure of extensional continuous functionals (introduced in Kleene (1959a) as 'countable functionals', and in Kreisel (1959)) as the standard model. Another model is obtained by taking the strongly majorizable functionals (introduced by Bezem (1985)), a variant of the heriditarily majorizable functionals from Howard (1973).

We can use the $\boldsymbol{\lambda}\mathbf{T}$-model $\mathfrak{M}(\mathbb{N})$ and the $\boldsymbol{\lambda}\mathbf{2T}$-model $\mathfrak{P}(\mathbb{N})$ also for the extended theories. This will be verified now.

4.3.1. THEOREM. $\mathfrak{M}(\mathbb{N})$ *is a model of* $\boldsymbol{\lambda}\mathbf{T}_{\mathbf{A}}$.

PROOF. The proof of Corollary 3.2.8 (ii) can be extended as follows. The rules (SE) and ($\perp$R) are sound since $[\![\boldsymbol{S}]\!]$ is the successor function and $\mathfrak{M}(\mathbb{N})$ is an ω-model.

As to the induction rule, suppose

$$\mathfrak{M}(\mathbb{N}), \rho \models \varphi(\boldsymbol{0}, y) \tag{1}$$

and

$$\mathfrak{M}(\mathbb{N}), \rho \models \varphi(x, Fy) \;\Rightarrow\; \mathfrak{M}(\mathbb{N}), \rho \models \varphi(\boldsymbol{S}x, y) \tag{2}$$

in order to show

$$\mathfrak{M}(\mathbb{N}), \rho \models \varphi(x, Q).$$

Now reason in $\mathfrak{M}(\mathbb{N})$. Let $n \in \mathbb{N}$. For each $k \leq n$ one has

$$\varphi(n-k, F^k(y)) \;\Rightarrow\; \varphi(n, y)$$

by (2) and induction on k. Hence (choosing $k = n$)

$$\varphi(0, F^k(y)) \;\Rightarrow\; \varphi(n, y).$$

Now by (1) we are done.

Finally, the propositional rules (P) are obviously sound. □

4.3.2. THEOREM. $\mathfrak{P}(\mathbb{N})$ *is a model of* $\boldsymbol{\lambda 2}\mathbf{T}_{\mathbf{A}}$.

PROOF. By extension of the proof of Theorem 3.6.7 with the following observations. The rules in (N) are satisfied since $\mathrm{dom}[\mathbf{0}] \cong \mathbb{N}$ and $\mathbf{S}^+$ represents the 'real' successor function; the rule (IR) is verified as in the proof above. Moreover, the propositional extension is obviously sound. □

By a similar argument one can show the following.

4.3.3. THEOREM. HEO2 *is a model of* $\boldsymbol{\lambda 2}\mathbf{T}_{\mathbf{A}}$.

It is convenient to formulate a notion of ω-model for arbitrary models of theories containing $\boldsymbol{\lambda}\mathbf{T}$

4.3.4. DEFINITION. Let $\mathfrak{M}$ be a first-order (or second-order) model of (an extension of) $\boldsymbol{\lambda}\mathbf{T}$. Then $\mathfrak{M}$ is said to be an *ω-model* if

$$\langle [\![\mathbf{0}]\!],\ [\![\boldsymbol{S}]\!],\ [\![\boldsymbol{0}]\!] \rangle \cong \langle \mathbb{N},\ S,\ 0 \rangle.$$

Obviously, the models $\mathfrak{M}(\mathbb{N})$, $\mathfrak{P}(\mathbb{N})$ and HEO2 are ω-models.

4.4. Realizability of Functional Specifications

The metamathematical results from Spector (1962) and Girard (1972) imply that $\lambda\mathbf{TB}$ and $\lambda\mathbf{2T}$ are equally powerful with respect to definability of functions on the natural numbers: the definable functions (on the numerals of type $\mathbf{0}$) are exactly those that are provably total in analysis.

It is not obvious how to compare the above theories with respect to definability in higher types. In the above situation it is clear what the domain of the functions in question should be: the set of natural numbers. Due to the variety of models (containing at type $\mathbf{1}$, e.g., all numerical functions ($\mathfrak{M}(\mathbb{N})$), or just the computable ones (HEO)) it is not clear how to specify a class of type-$\mathbf{2}$ functionals for investigation of definability.

Here we choose for a *syntactical* specification of functions and functionals in the common language of $\lambda\mathbf{TB}$ and $\lambda\mathbf{2T}$, namely the language of $\lambda\mathbf{T}$. The notion of definability is replaced by the concept of realizability of $\lambda\mathbf{T}$-specifications, defined in *semantical* terms

We will show that the realizability concept is not vacuous: every computable function on the natural numbers can be specified in $\lambda\mathbf{T}$. Then we reformulate the above result about definability of functions in terms of realizability of specifications.

4.4.1. DEFINITION. Let $\sigma \in \mathbb{T}_1$. A *σ-specification* is a formula $\Sigma^\sigma \equiv \Sigma(f^\sigma)$ in the language of $\lambda\mathbf{T}_\mathbf{A}$, containing one free variable of type σ (the *main variable*) and possibly containing other free variables of lower types (the *auxiliary variables*). We write $\Sigma^\sigma \equiv \Sigma(f^\sigma)$, or $\Sigma(f^\sigma, \vec{x})$ if we want to display the auxiliary variables explicitly.

4.4.2. DEFINITION. (i) Let ψ be a formula. By $\lambda\Box \models_\omega \psi$ we denote that ψ is *true in all ω-models* of $\lambda\Box$.

(ii) Let $\lambda\Box$ be an extension of $\lambda\mathbf{T}$, and let $\Sigma(f^\sigma, \vec{x})$ be a σ-specification. Then Σ is said to be *realizable* in $\lambda\Box$ if there exists $F \in \mathrm{Term}_\sigma(\lambda\Box)$ such that

$$\lambda\Box \models_\omega \Sigma(F, \vec{x}),$$

i.e. $\forall \vec{x}\, \Sigma(F, \vec{x})$ holds in all ω-models of $\lambda\Box$.

The following is the traditional syntactical notion of definability at type $\mathbf{1}$.

4.4.3. DEFINITION. Let $F \in \mathrm{Term}_\mathbf{1}(\lambda\Box)$, and $f : \mathbb{N} \to \mathbb{N}$. Then F is said to *$\lambda\Box$-define* f if for all $n \in \mathbb{N}$

$$\lambda\Box \vdash F\overline{n} =_\mathbf{0} \overline{f(n)}.$$

4.4.4. PROPOSITION. *Each computable function f on $\mathbb{N}$ has a $\mathbf{1}$-specification, i.e. a specification Σ that is satisfied only by type-$\mathbf{1}$ objects behaving like f.*

PROOF. Let T be the computation predicate and U the result extracting function from Kleene's Normal Form Theorem, such that

$$\{e\}(x) \simeq U(\mu z.T(e,x,z)).$$

Remember that T and U are primitive recursive and hence expressible in $\boldsymbol{\lambda}\mathbf{T}$. Now let $f : \mathbb{N} \to \mathbb{N}$ be computable, say with index e. Now take

$$\begin{aligned} \Sigma^1_e(f^1) &\equiv T(\bar{e},x,z) \to f^1 x =_0 Uz \\ &\equiv \Sigma^1_e(f^1,x^0,z^0). \end{aligned}$$

Obviously, this completely caputures the behaviour of f. □

In particular, F is a term realizing the above Σ in $\boldsymbol{\lambda}\Box$ (i.e. $\boldsymbol{\lambda}\Box \models_\omega \Sigma_e(F,x,z)$) iff F $\boldsymbol{\lambda}\Box$-defines f.

4.4.5. DEFINITION. Let $\boldsymbol{\lambda}\Box$ and $\boldsymbol{\lambda}\boxtimes$ be theories.

(i) By $\boldsymbol{\lambda}\Box \preccurlyeq^\sigma \boldsymbol{\lambda}\boxtimes$ we denote that every $\boldsymbol{\lambda}\Box$-realizable σ-specification is realizable in $\boldsymbol{\lambda}\boxtimes$.

(ii) $\boldsymbol{\lambda}\Box$ and $\boldsymbol{\lambda}\boxtimes$ are *realization equivalent at type* σ (notation $\boldsymbol{\lambda}\Box \approx^\sigma \boldsymbol{\lambda}\boxtimes$) if both $\boldsymbol{\lambda}\Box \preccurlyeq^\sigma \boldsymbol{\lambda}\boxtimes$ and $\boldsymbol{\lambda}\boxtimes \preccurlyeq^\sigma \boldsymbol{\lambda}\Box$.

Now we can translate the results from Spector (1962) and Girard (1972) into a realization property. First we need some technicalities.

4.4.6. DEFINITION. Let $f : \mathbb{N} \to \mathbb{N}$. Then $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f}$ is the theory obtained by extending $\boldsymbol{\lambda}\mathbf{T}$ with a constant $\boldsymbol{f}$ and axioms

$$\boldsymbol{f}\,\bar{n} = \overline{f(n)}. \qquad (f)$$

We use $\to$ (and $\twoheadrightarrow$) to refer to the (multistep) *reduction relations* corresponding to the equalities in the theories $\boldsymbol{\lambda}\mathbf{T}$ and $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f}$.

4.4.7. LEMMA. (i) $\to$ *is Church-Rosser on* $\mathrm{Term}(\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f})$.

(ii) $\to$ *is strongly normalizing.*

(iii) *Each closed* $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f}$*-term of type* $\mathbf{0}$ *reduces to a unique numeral.*

PROOF. (i) By the observation that the contraction rules for $\boldsymbol{f}$ ($\boldsymbol{f}\,\bar{n} \to \overline{f(n)}$) define a notion of δ-reduction (see Barendregt (1984), Theorem 15.3.3) and the fact that $\boldsymbol{\lambda}\mathbf{T}$ is Church-Rosser (folklore).

(ii) Straightforward adaptation of Tait's computability argument (see Tait (1967), cf. Barendregt (1984), Theorem A.2.3).

(iii) We show that every closed normal form P of type $\mathbf{0}$ is a numeral. Then we are done by (i) and (ii). We proceed by induction on the length of normal forms. Note that every normal form has the shape $\lambda\vec{x}.aM_1 \cdots M_k$, with a atomic, i.e. a variable or a constant, and the M_i again in normal form. Since P has type $\mathbf{0}$ the abstraction $\lambda\vec{x}$ is empty, and a is a constant.

If $a \equiv \boldsymbol{0}$ then we are done.

If $a \equiv \boldsymbol{S}$ then $P \equiv \boldsymbol{S}Q$ for some closed normal form Q of type $\mathbf{0}$, so the induction hypothesis applies.

If $a \equiv \boldsymbol{f}$ then $P \equiv \boldsymbol{f}Q$ for some Q. But then P is a redex by induction hypothesis, contradiction.

If $a \equiv \boldsymbol{R}_\sigma$ then $\sigma \equiv \mathbf{0}$ and $P \equiv \boldsymbol{R}_{\mathbf{0}}MNQ$ with Q closed and of type $\mathbf{0}$. Then again P would be a redex, contradiction. □

4.4.8. LEMMA. *Let $F \in \mathrm{Term}_1(\boldsymbol{\lambda}\Box)$, and $f : \mathbb{N} \to \mathbb{N}$. Suppose f is $\boldsymbol{\lambda}\Box$-defined by F. Let $\Sigma^{\mathbf{1}} \equiv \Sigma(f^{\mathbf{1}}, \vec{x}^{\mathbf{0}})$ be a $\mathbf{1}$-specification. Then*

$$\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f} \models_\omega \forall\vec{x}\, \Sigma(\boldsymbol{f}, \vec{x}) \;\Leftrightarrow\; \boldsymbol{\lambda}\Box \models_\omega \forall\vec{x}\, \Sigma(F, \vec{x}).$$

PROOF. Note that each ω-model of $\boldsymbol{\lambda}\Box$ is a model of $\boldsymbol{\lambda}\Box_{\mathbf{A}}$. By Theorem 4 2.11, we can write $\Sigma^{\mathbf{1}}(f, \vec{x})$ as $Gf\vec{x} =_{\mathbf{0}} \bar{1}$ for some closed $\boldsymbol{\lambda}\mathbf{T}$-term G. Assume F $\boldsymbol{\lambda}\Box$-defines f. Observe that each ω-model of $\boldsymbol{\lambda}\Box$ is also a model of $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f}$, with $[\![\boldsymbol{f}]\!] = [\![F]\!]$ ('$= f$') by extensionality.

($\Rightarrow$) Suppose $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f} \models_\omega \forall\vec{x}\, \Sigma(\boldsymbol{f}, \vec{x})$. Then for all $\vec{n} \in \mathbb{N}$

$$\begin{aligned} [\![GF\bar{\vec{n}}]\!] &= [\![G]\!]\,[\![F]\!]\,[\![\bar{\vec{n}}]\!] \\ &= [\![G]\!]\,[\![\boldsymbol{f}]\!]\,[\![\bar{\vec{n}}]\!] \\ &= [\![\bar{1}]\!] \end{aligned}$$

and we are done by the above observation.

($\Leftarrow$) Suppose $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f} \not\models_\omega \forall\vec{x}\, \Sigma(\boldsymbol{f}, \vec{x})$. Then $[\![G\boldsymbol{f}\bar{\vec{n}}]\!] = [\![\bar{p}]\!]$ in some ω-model of $\boldsymbol{\lambda}\mathbf{T}\boldsymbol{f}$, for some $\vec{n}, p \in \mathbb{N}$ with $p \neq 1$. Note that $G\boldsymbol{f}\vec{n}$ is a closed term of type $\mathbf{0}$. Therefore $G\boldsymbol{f}\vec{n} \twoheadrightarrow \bar{p}$ by Lemma 4.4.7 (iii). This reduction sequence can be transformed into a $\boldsymbol{\lambda}\Box$-derivation of $GF\bar{\vec{n}} =_{\mathbf{0}} \bar{p}$, using a subsidiary derivation of $Fk =_{\mathbf{0}} \overline{f(k)}$ for each reduction step $\boldsymbol{f}\,\bar{k} \to \overline{f(k)}$. Hence $[\![GF\bar{\vec{n}}]\!] \neq [\![\bar{1}]\!]$, so $\boldsymbol{\lambda}\Box \not\models_\omega \forall\vec{x}\, \Sigma(F, \vec{x})$. □

4.4.9. THEOREM. $\boldsymbol{\lambda}\mathbf{T}\mathbf{B}_{\mathbf{A}} \approx^{\mathbf{1}} \boldsymbol{\lambda}\mathbf{2}\mathbf{T}_{\mathbf{A}}$.

PROOF. By Lemma 4.4.8, using the fact that $\boldsymbol{\lambda}\mathbf{T}\mathbf{B}_{\mathbf{A}}$ and $\boldsymbol{\lambda}\mathbf{2}\mathbf{T}_{\mathbf{A}}$ define the same f's, by the results of Spector (1962) and Girard (1972). □

4.5. First-Order Non-Conservativity

In this section we will consider a **3**-specification $\Sigma \equiv \Sigma(\Phi^{\mathbf{3}})$ and show that this specification is realizable in $\lambda\mathbf{TB_A}$, but not in $\lambda\mathbf{T_A}$. (In the next section we will show that even in $\lambda\mathbf{2T}$ the specification is not realizable.)

The positive result for $\lambda\mathbf{TB_A}$ will be proved by a purely syntactical method. With a little extra effort one can use the specification Σ to construct a formula ψ in the language of $\lambda\mathbf{T_A}$ such that

$$\lambda\mathbf{TB_A} \vdash \psi,$$
$$\lambda\mathbf{T_A} \nvdash \psi,$$

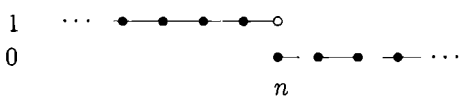
thus showing that $\lambda\mathbf{TB_A}$ is a non-conservative extension of $\lambda\mathbf{T_A}$. This does not really come as a surprise since it follows from Spector (1962) that in the system $\lambda\mathbf{TB_A}$ the consistency of Peano Arithmetic, i.e. $\neg\mathrm{Proof}_{\mathbf{PA}}(x, \ulcorner 0 = 1 \urcorner)$, can be proved (after suitable encoding).

The formula presented here, however, does not refer to any metamathematical properties and can be formulated in a simple way.

4.5.1. DEFINITION. The term $\Delta \in \mathrm{Term}_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}(\lambda\mathbf{T})$ is defined by

$$\Delta \equiv \lambda xy.\mathsf{if}\ y < x\ \mathsf{then}\ \bar{1}\ \mathsf{else}\ \bar{0}.$$

The intuition is that Δn is a so-called *stepfunction* that can be depicted as follows.



4.5.2. DEFINITION. The **3**-specification $\Sigma \equiv \Sigma(\Phi)$, with Φ a type-**3** variable, is defined by

$$\Sigma(\Phi) \equiv [x < \Phi\varphi \to \varphi(\Delta x) \geq x \quad \wedge \quad \varphi(\Delta(\Phi\varphi)) < \Phi\varphi].$$

Here φ is a type-**2** variable and x a type-**0** variable. Note that $\Sigma(\Phi)$ can be written in the form $G\Phi\varphi x = \bar{0}$ or $G\Phi = \lambda\varphi x.\bar{0}$

4.5.3. INTUITION. If Φ satisfies $\Sigma(\Phi)$ then Φ performs the following minimization.

$$\Phi\varphi \simeq \mu x.\ \varphi(\Delta x) < x.$$

This minimization is only well defined for φ such that there exists x with $\varphi(\Delta x) < x$. Since Σ will be shown to have a bar recursive solution Φ, the minimization

is well defined for φ taken from a model of bar recursion, such as the countable/continuous functionals, or the (strongly) majorizable functionals. A counter model against $\mathbf{\Sigma}(\Phi)$ must contain a φ for which the minimization is not well defined. Since every continuous functional of type **2** is also majorizable, any counter model must contain a non-majorizable functional φ. This is not very problematic for counter models satisfying $\boldsymbol{\lambda}\mathbf{T_A}$ but turned out to be difficult for counter models satisfying $\boldsymbol{\lambda}\mathbf{2T_A}$, since all 'standard' models of $\boldsymbol{\lambda}\mathbf{2T_A}$ seem to exhibit an inherent continuity.

4.5.4. DEFINITION. Let $\boldsymbol{\lambda}\Box$, $\boldsymbol{\lambda}\boxtimes$ be two of our systems. We say that $\mathbf{\Sigma}$ is $\boldsymbol{\lambda}\Box$-*solvable* in $\boldsymbol{\lambda}\boxtimes$ if for some $\Phi \in \mathrm{Term}_3(\boldsymbol{\lambda}\Box)$ one has

$$\boldsymbol{\lambda}\boxtimes \vdash \mathbf{\Sigma}(\Phi).$$

Obviously, if $\mathbf{\Sigma}$ is $\boldsymbol{\lambda}\Box$-solvable in $\boldsymbol{\lambda}\Box$ then $\mathbf{\Sigma}$ is realizable in $\boldsymbol{\lambda}\Box$. We first show that $\mathbf{\Sigma}$ is not realizable in $\boldsymbol{\lambda}\mathbf{T_A}$ (so *a fortiori* $\mathbf{\Sigma}$ is not $\boldsymbol{\lambda}\mathbf{T}$-solvable in $\boldsymbol{\lambda}\mathbf{T_A}$)

4.5.5. PROPOSITION. *There exists no* $\Phi \in \mathrm{Term}_3(\boldsymbol{\lambda}\mathbf{T})$ *such that*

$$\boldsymbol{\lambda}\mathbf{T_A} \models_\omega \mathbf{\Sigma}(\Phi).$$

PROOF. Suppose, towards a contradiction, $\boldsymbol{\lambda}\mathbf{T_A} \models_\omega \mathbf{\Sigma}(\Phi)$ with $\Phi \in \mathrm{Term}_3(\boldsymbol{\lambda}\mathbf{T})$. Consider the ω-model $\mathfrak{M}(\mathbb{N})$. Define $\delta \in \mathbb{N}_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ by

$$\begin{aligned} \delta(n)(x) &= 1 \quad \text{if } x < n, \\ &= 0 \quad \text{if } x \geq n. \end{aligned}$$

Note that $[\![\mathbf{\Delta}]\!] = \delta$; denotation: $\delta_n = \delta(n)$. Take $\varphi \in \mathbb{N}_{\mathbf{2}}$ such that for each $n \in \mathbb{N}$

$$\varphi(\delta_n) = n.$$

Then one has

$$\varphi(\delta([\![\Phi]\!]\varphi)) = [\![\Phi]\!]\varphi,$$

contradicting $\mathfrak{M}(\mathbb{N}) \models \mathbf{\Sigma}(\Phi)$. $\square$

It can easily be seen that $\mathbf{\Sigma}$ is $\boldsymbol{\lambda}\mathbf{TB}$-solvable in $\boldsymbol{\lambda}\mathbf{TB_A}$. Indeed, the minimization stated above can be obtained by constructing a 'searching functional' Ψ of type $\mathbf{2}\to\mathbf{0}\to\mathbf{0}$ such that

$$\begin{aligned} \Psi\varphi s &= 0 && \text{if } \varphi[s] < \mathrm{lh}(s), \\ &= 1 + \Psi\varphi(s * \langle 1 \rangle) && \text{otherwise,} \end{aligned}$$

and finally defining $\Phi \equiv \lambda\varphi.\Psi\varphi\langle\,\rangle$. From the form of the equations for Ψ it can be expected that such a Ψ can be constructed using the bar recursor.

4 5 6 DEFINITION The term $\mathbf{F_B} \in \mathrm{Term}_3(\boldsymbol{\lambda}\mathbf{TB})$ is defined as follows

$$\mathbf{F_B} \equiv \lambda\varphi\, \boldsymbol{B}\varphi GH\langle\rangle,$$

where

$$\begin{aligned} G &\equiv \lambda\sigma\, \bar{0}, \\ H &\equiv \lambda sf\, \boldsymbol{S}(f\bar{1}) \end{aligned}$$

In order to show that $\mathbf{F_B}$ is a solution for $\boldsymbol{\Sigma}$ we reason informally in $\boldsymbol{\lambda}\mathbf{TB_A}$, the argument can easily be formalized

4 5 7 DEFINITION Let $P(x, \varphi)$ abbreviate

$$\begin{aligned} x < \mathbf{F_B}\varphi \;\rightarrow\; & (\varphi(\boldsymbol{\Delta}x) \geq x \;\wedge \\ & \mathbf{F_B}\varphi = x + 1 + \boldsymbol{B}\varphi GH1^{x+1}), \end{aligned}$$

where $1^x = \underbrace{\langle 1, 1, \quad, 1\rangle}_{x \text{ times}}$

4 5 8 LEMMA $\boldsymbol{\lambda}\mathbf{TB_A} \vdash P(x, \varphi)$

PROOF By induction on x Note that $[1^x] = \boldsymbol{\Delta}x$, by extensionality

Basis $P(0, \varphi)$ holds since trivially $\varphi(\boldsymbol{\Delta}0) \geq 0$ and $\mathbf{F_B}\varphi > 0$ implies $\varphi[\langle\rangle] \geq 0 = \mathrm{lh}(\langle\rangle)$ and therefore

$$\begin{aligned} \mathbf{F_B}\varphi &= H\langle\rangle(\lambda y\, \boldsymbol{B}\varphi GH\langle y\rangle) \\ &= 1 + \boldsymbol{B}\varphi GH\langle 1\rangle \end{aligned}$$

Induction step Suppose $P(x, \varphi)$ in order to show $P(x + 1, \varphi)$ Suppose $x + 1 < \mathbf{F_B}\varphi$ Then $x < \mathbf{F_B}\varphi$ so by induction hypothesis

$$\mathbf{F_B}\varphi = x + 1 + \boldsymbol{B}\varphi GH1^{x+1}$$

We claim that $\varphi(\boldsymbol{\Delta}(x+1)) \geq x+1$ Indeed, suppose $\varphi(\boldsymbol{\Delta}(x+1)) < x+1$ Then $\varphi[1^{x+1}] < x + 1$, so $\boldsymbol{B}\varphi GH1^{x+1} = 0$, so $\mathbf{F_B}\varphi = x + 1$, contradicting $\mathbf{F_B}\varphi > x + 1$ Now it follows that

$$\begin{aligned} \boldsymbol{B}\varphi GH1^{x+1} &= H1^{x+1}(\lambda y\, \boldsymbol{B}\varphi GH(1^{x+1} * \langle y\rangle)) \\ &= 1 + \boldsymbol{B}\varphi GH1^{x+2}, \end{aligned}$$

so

$$\mathbf{F_B}\varphi = x + 2 + \boldsymbol{B}\varphi GH1^{x+2},$$

which completes the induction step □

4 5 9 PROPOSITION $\boldsymbol{\lambda TB_A} \vdash \boldsymbol{\Sigma(F_B)}$

PROOF Observe that $\mathbf{F_B}\varphi > 0$ By Lemma 4 5 8 one has

$$P(\mathbf{F_B}\varphi - 1, \varphi),$$

and therefore

$$\mathbf{F_B}\varphi = \mathbf{F_B}\varphi + \boldsymbol{B}\varphi GH1^{\mathbf{F_B}\varphi},$$

so

$$\boldsymbol{B}\varphi GH1^{\mathbf{F_B}\varphi} = 0,$$

which implies

$$\begin{aligned} \varphi(\boldsymbol{\Delta}(\mathbf{F_B}\varphi)) &= \varphi[1^{\mathbf{F_B}\varphi}] \\ &< \mathrm{lh}(1^{\mathbf{F_B}\varphi}) \\ &= \mathbf{F_B}\varphi \end{aligned}$$

Combining this with $P(x, \varphi)$ gives

$$\boldsymbol{\Sigma(F_B)} \quad \square$$

4 5 10 COROLLARY $\boldsymbol{\lambda TB_A} \not\leq^3 \boldsymbol{\lambda T_A}$

PROOF. By the propositions 4 5 5 and 4 5 9 $\square$

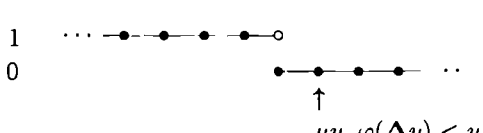
The above corollary is just preparing for the results in the next section In fact one already has $\boldsymbol{\lambda TB_A} \not\leq^1 \boldsymbol{\lambda T_A}$, by the metamathematical results of Spector (1962) and Godel (1958)

It will turn out that $\boldsymbol{\Sigma}$ is even $\boldsymbol{\lambda T}$-solvable in $\boldsymbol{\lambda TB_A}$ Use is made of a trick due to G Kreisel, also employed by J A Bergstra and J Terlouw, see Barendregt (1984), p 581

4 5 11 DEFINITION Define $f_\varphi \in \mathrm{Term}_1(\boldsymbol{\lambda T})$ by

$$f_\varphi \equiv \lambda x \text{ if } \forall y \leq x+1 \ \ \varphi(\boldsymbol{\Delta}y) \geq y \text{ then } \mathbf{1} \text{ else } \bar{0}$$

4 5 12 INTUITION f_φ is a stepfunction which looks as follows



So $f_\varphi = \mathbf{\Delta}((\mu y\ \ \varphi(\mathbf{\Delta}y) < y) - 1)$ (provided $\exists y\ \varphi(\mathbf{\Delta}y) < y$, otherwise '$f_\varphi = \mathbf{\Delta}_\infty$')
Therefore

$$\varphi(f_\varphi) \geq (\mu y\ \ \varphi(\mathbf{\Delta}y) < y) - 1$$

and hence

$$(\mu y\ \ \varphi(\mathbf{\Delta}y) < y)\ \leq\ \varphi(f_\varphi) + 1$$

This gives a primitive recursive upperbound of the minimization expressed in $\mathbf{\Sigma}$

4 5 13 DEFINITION The term $\mathbf{F}_\mathbf{T} \in \mathrm{Term}_3(\boldsymbol{\lambda}\mathbf{T})$ is defined as follows

$$\mathbf{F}_\mathbf{T} \equiv \lambda\varphi\ \ \mu x{\leq}\varphi(f_\varphi){+}1\ \ \varphi(\mathbf{\Delta}x) < x$$

4 5 14 PROPOSITION $\boldsymbol{\lambda}\mathbf{TB_A} \vdash \mathbf{F}_\mathbf{T} = \mathbf{F}_\mathbf{B}$

PROOF Again we reason informally in $\boldsymbol{\lambda}\mathbf{TB_A}$ From Proposition 4 5 9 we know $\mathbf{\Sigma}(\mathbf{F}_\mathbf{B})$, so

$$x < \mathbf{F}_\mathbf{B}\varphi \to \varphi(\mathbf{\Delta}x) \geq x, \tag{1}$$

$$\varphi(\mathbf{\Delta}(\mathbf{F}_\mathbf{B}\varphi)) < \mathbf{F}_\mathbf{B}\varphi \tag{2}$$

From (1) it follows that

$$\forall y \leq \mathbf{F}_\mathbf{B}\varphi - 1 \qquad \varphi(\mathbf{\Delta}y) \geq y \tag{3}$$

and therefore

$$x < \mathbf{F}_\mathbf{B}\varphi - 1 \to f_\varphi x = 1, \tag{4}$$

and

$$x \geq \mathbf{F}_\mathbf{B}\varphi - 1 \to f_\varphi x = 0 \tag{5}$$

From (4) and (5) one obtains by extensionality

$$f_\varphi = \mathbf{\Delta}(\mathbf{F}_\mathbf{B}\varphi - 1)$$

Hence again by (1)

$$\varphi(f_\varphi) \geq \mathbf{F}_\mathbf{B}\varphi - 1$$

and therefore

$$\mathbf{F}_\mathbf{B}\varphi \leq \varphi(f_\varphi) + 1$$

Combining this with (2) and again (1) (ensuring the minimality of $\mathbf{F}_\mathbf{B}\varphi$ with respect to $\varphi(\mathbf{\Delta}x) < x$) yields

$$\mathbf{F}_\mathbf{T}\varphi = \mathbf{F}_\mathbf{B}\varphi$$

and hence by extensionality

$$\mathbf{F}_\mathbf{T} = \mathbf{F}_\mathbf{B}\ \ \square$$

4 5 15 COROLLARY $\boldsymbol{\lambda}\mathbf{TB_A} \vdash \mathbf{\Sigma}(\mathbf{F}_\mathbf{T})$

PROOF By the propositions 4 5 9 and 4 5 14 □

Now we can formulate the non conservativity result obtained in this section

4 5 16 THEOREM $\lambda\mathbf{TB_A}$ *is a non conservative extension of* $\lambda\mathbf{T_A}$

PROOF Note that $\Sigma(\mathbf{F_T})$ is an equation in the language of $\lambda\mathbf{T_A}$ By Proposition 4 5 5

$$\lambda\mathbf{T_A} \not\vdash \Sigma(\mathbf{F_T}),$$

whereas by Proposition 4 5 15

$$\lambda\mathbf{TB_A} \vdash \Sigma(\mathbf{F_T}) \quad \square$$

It is important to stress that our results have little to do with proof theoretic strength in the usual sense Luckhardt (1975) presents a system $\lambda\mathbf{T} + \mu$ which has the same proof theoretic strength as Heyting's Arithmetic Our specification Σ is in fact a special case of the defining equation for μ It is not difficult to prove that $\vdash_{\lambda\mathbf{T}+\mu} \Sigma(\mathbf{F_T})$ As a consequence we can complement the results from Luckhardt (1975) with the following theorem

4 5 17 THEOREM $\lambda\mathbf{T} + \mu$ *is a non conservative extension of* $\lambda\mathbf{T_A}$

Similar results can be obtained for the extension of $\lambda\mathbf{T}$ with a modulus of uniform continuity (a so-called *fan functional*)

4.6. Non-Realizability in Second-Order Lambda Calculus

In this section it will be shown that Σ is not even realizable in $\lambda\mathbf{2T_A}$ This suggests that bar recursion is, in some sense, a more powerful extension of $\lambda\mathbf{T}$ than the concept of polymorphism (see the discussion in the introduction)

The idea of the proof is the same as the proof for $\lambda\mathbf{T_A}$ (Proposition 4 5 5), but the implementation is far more difficult The presence of a type-**2** functional like φ, introducing a 'fatal discontinuity', would solve the problem

A first attempt would be to extend Kleene's partial applicative structure with codes for an oracle function, thus obtaining a relative version of the model HEO2 A naive solution like adding an oracle φ such that

$$\begin{aligned} \varphi(e) &= n \quad \text{if } \{e\} = \delta_n, \\ &= 0 \quad \text{otherwise} \end{aligned}$$

fails since an index of such a φ will get lost in the PER-construction (Let $\{\ \}^\varphi$ denote the relativized version of $\{\ \}$ There will be e_1, e_2 such that $\{e_1\} = \delta_n$ and $\{e_2\} \neq \delta_n$ for some $n > 0$ but $\{e_1\}^\varphi = \{e_2\}^\varphi$, φ acts differently on e_1 and e_2) This is not just a technical accident but has a fundamental reason one can show that

the Kreisel-Lacombe-Schoenfield theorem (see Kreisel et al (1959)), expressing that all type-**2** elements of HEO2 are continuous, can be relativized Hence every recursion theoretic oracle is doomed to fail This indicates the necessity of 'impredicative oracles' Other well known model constructions for $\boldsymbol{\lambda 2}$ and $\boldsymbol{\lambda 2T}$ seem to exhibit an inherent continuity

Using the construction of Section 3 6 we can prove an analogue of Proposition 4 5 5

4 6 1 PROPOSITION *There exists no* $\Phi \in \mathrm{Term}_3(\boldsymbol{\lambda 2T})$ *such that*

$$\boldsymbol{\lambda 2T_A} \models_\omega \Sigma(\Phi)$$

PROOF Suppose $\boldsymbol{\lambda 2T_A} \models_\omega \Sigma(\Phi)$ We want to derive a contradiction We consider the model $\mathfrak{P}(\mathbb{N})$ Take $\varphi \quad \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ such that $\varphi(\delta_n) = n$ (like in the proof of Proposition 4 5 5) Construct $\mathbf{D} \in \Lambda^\circ\mathfrak{M}$ such that

$$\begin{aligned} \mathbf{D}\ulcorner n \urcorner\ulcorner m \urcorner &= \ulcorner 1 \urcorner \quad \text{if } m < n, \\ &= \ulcorner 0 \urcorner \quad \text{otherwise} \end{aligned}$$

Note that $\langle\!\langle \mathbf{D} \rangle\!\rangle \in \mathrm{dom}[\mathbf{0}{\to}\mathbf{0}{\to}\mathbf{0}]_\xi$, more specifically, $\langle\!\langle \mathbf{D} \rangle\!\rangle \in [\![\boldsymbol{\Delta}]\!]$ Moreover, by the Classification Theorem 3 4 8 (1) one has $\langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle \in \mathrm{dom}[(\mathbf{0}{\to}\mathbf{0}){\to}\mathbf{0}]_\xi$, and

$$\langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle(\langle\!\langle \mathbf{D} \rangle\!\rangle([\Phi]\langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle)) = [\Phi]\langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle$$

This implies

$$[\varphi(\boldsymbol{\Delta}(\Phi))]_{\rho(\varphi = \langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle)}\ [\mathbf{0}]\ [\Phi\varphi]_{\rho(\varphi = \langle\!\langle \boldsymbol{\varphi} \rangle\!\rangle)},$$

contradicting $\mathfrak{P}(\mathbb{N}) \models \Sigma(\Phi)$ Note that the Isomorphism Theorem 3 4 9 guarantees that distinct natural numbers correspond to distinct model elements of type 0 □

4 6 2 COROLLARY $\boldsymbol{\lambda TB_A} \not\preccurlyeq^3 \boldsymbol{\lambda 2T_A}$

It is open whether $\boldsymbol{\lambda 2T_A} \preccurlyeq^3 \boldsymbol{\lambda TB_A}$ holds Moreover the type **2** case is still unsolved

The original proof of Proposition 4 6 1 in Barendsen and Bezem (1991) used essentially the same technique but in a more restricted way (aiming just at the non-realizability result above) This involved a per model over untyped λ-calculus with one single oracle (namely for the above φ), with a reduction relation $\to_\varphi$ such that

$$\varphi F \to_\varphi \ulcorner \varphi(f) \urcorner \qquad \text{if} \qquad F \rhd_{\beta\varphi} f$$

See Barendsen and Bezem (1992) for details

4.7. Extensions to Higher-Order Lambda Calculus

It is possible to extend our results to higher-order lambda calculi. In these ω-order systems one considers *type constructors*. These include all types but also functions from types to types, like $\lambda\alpha\ \alpha{\to}\alpha$, functionals on these, and so on. In this section we briefly describe the systems and our results.

The (extensions of) the ω-order system $\boldsymbol{\lambda\omega}$ can be described using the notion of *kind*. Kinds are the 'types of constructors'. First choose a constant $*$, the intended meaning of $*$ is the collection of all types. Now the kinds of $\boldsymbol{\lambda\omega}$ are defined by the following abstract syntax.

$$\mathbb{K} = * \mid \mathbb{K}{\to}\mathbb{K}$$

Notice the similarity with the *types* of $\boldsymbol{\lambda}^{\tau}$.

For each $\kappa \in \mathbb{K}$, the *constructors* of kind κ (notation Constr_κ) are defined as follows. One easily recognizes the types in $\mathbb{T}_2$ as a subcollection of Constr_*. For each $\kappa \in \mathbb{K}$, let CVar_κ be a set of *constructor variables*.

$$\begin{aligned}
\alpha^\kappa \in \mathrm{CVar}_\kappa &\Rightarrow \alpha^\kappa \in \mathrm{Constr}_\kappa,\\
C \in \mathbb{C} &\Rightarrow C \in \mathrm{Constr}_*,\\
\sigma, \tau \in \mathrm{Constr}_* &\Rightarrow (\sigma{\to}\tau) \in \mathrm{Constr}_*,\\
\sigma \in \mathrm{Constr}_*,\ \alpha^\kappa \in \mathrm{CVar}_\kappa &\Rightarrow (\forall\alpha^\kappa\ \sigma) \in \mathrm{Constr}_*\\
\gamma \in \mathrm{Constr}_{\kappa_1\to\kappa_2},\ \delta \in \mathrm{Constr}_{\kappa_1} &\Rightarrow (\gamma\delta) \in \mathrm{Constr}_{\kappa_2}\\
\alpha^{\kappa_1} \in \mathrm{CVar}_{\kappa_1},\ \gamma \in \mathrm{Constr}_{\kappa_2} &\Rightarrow (\lambda\alpha^{\kappa_1}\ \gamma) \in \mathrm{Constr}_{\kappa_1\to\kappa_2}
\end{aligned}$$

Then, e.g., $\lambda\alpha^*\ \alpha{\to}\alpha \in \mathrm{Constr}_{*\to *}$.

Let $\boldsymbol{\lambda}\square$ range over the ω-order systems. Now the collection of *terms* of $\boldsymbol{\lambda}\square$ (in habitants of types $\sigma \in \mathrm{Constr}_*$) can be defined. For a proper syntactic treatment one is forced to consider variable *contexts* instead of annotated term variables. We refrain from going into the details of this. The system $\boldsymbol{\lambda\omega}$ is the plain ω-order calculus (without term constants), and $\boldsymbol{\lambda\omega T}$ is the ω-order variant of $\boldsymbol{\lambda T}$.

The per semantics of the second order systems (in a per structure $\mathfrak{P} = (A, \ , \mathcal{T})$) can be extended to ω order systems, as follows. The higher order constructors are interpreted in the full type structure over $\mathrm{PER}(A)$, by setting

$$\begin{aligned}
[*] &= \mathrm{PER}(A),\\
[\kappa_1{\to}\kappa_2] &= [\kappa_2]^{[\kappa_1]}
\end{aligned}$$

Moreover, elements of Constr_* are interpreted by extending the per interpretation of types into $\mathrm{PER}(A)$ described in Definition 3.3.6. In particular,

$$[\forall\alpha^\kappa\ \sigma]_\xi = \bigwedge_{F\in[\kappa]} [\sigma]_{\xi(\alpha^\kappa = F)}$$

Without proof we mention the following

4.7.1. THEOREM. (i) *Let* $\mathfrak{P}$ *be a per structure Then* $\mathfrak{P}$ *is a model of* $\boldsymbol{\lambda\omega}$.
(ii) $\mathfrak{P}(\mathbb{N})$ *is a model of* $\boldsymbol{\lambda\omega T_A}$.

4.7.2. COROLLARY. $\boldsymbol{\lambda TB_A} \not\approx^3 \boldsymbol{\lambda\omega T_A}$.

PROOF. Analogous to the proof of Corollary 4.6.2. □

Part II

Graph Rewrite Systems

Chapter 5

Conventional Typing in Graph Rewrite Systems

5.1. Introduction

In order to study fundamental aspects of programming languages one considers mathematical models of computation For imperative programming languages these are abstract machines, such as Turingmachines from Turing (1936/7)

For functional programming languages one considers models based on *rewriting*, like the lambda calculus introduced by Church (1936) Another model is provided by *term rewrite systems* Graph rewriting is a relatively new concept, providing a model that is sufficiently elegant and abstract, but at the same time incorporates mechanisms that are more realistic with respect to actual implementation techniques

A graph theoretic version of the λ-calculus has been introduced by Wadsworth (1971)

We consider a graph theoretic variant of term rewriting, introduced in Barendregt et al (1987b) The present work deals with a restricted form of GRS's the so called *term graph rewrite systems* (TGRS's, see Barendregt et al (1987a)) TGRS's are very well suited as a basis for (implementation of) functional languages, as is demonstrated by the graph rewrite language *Concurrent Clean*, see Nocker et al (1991)

The concept of typing in lambda calculus is well known To study the effect of *patterns* in function definitions on typing, a Curry-style type assignment system on (applicative) term rewriting systems has been developed by van Bakel et al (1992) The types in this system resemble those of the simply typed lambda calculus, the use of type schemes for graph symbols introduces a weak form of polymorphism Moreover arbitrary type constructors are incorporated

In the present paper the notion of type assignment to TRS's of van Bakel et al (1992) will be extended to general term graphs (i e graphs that are not necessarily trees) in a very natural way Some aspects of typing are even more

convenient in graph theoretical setting E g special contexts for variables are not necessary since multiple occurrences of the same variable are represented by one single node Moreover, the extra features of graph rewriting (shared c q cyclic objects) are treated without extra effort

As in van Bakel et al (1992), type assignment is not defined using a set of deduction rules but, more directly, by supplying nodes of the graphs and of the rewrite rules with types in a consistent way The consistency is expressed in a 'local' constraint for each node Graph symbols are supplied with a type by a so called *type environment* We show how to incorporate type constructors introduced by *algebraic type specifications*

5.2. Graph Rewriting

Term graph rewrite systems were introduced in Barendregt et al (1987a) This section summarizes some convenient (alternative) denotations and basic concepts, which are provided by Barendsen and Smetsers (1992) see also (1994)

Finite sequences

5 2 1 DEFINITION (i) A finite sequence (over A) is a tuple $s = (a_1\ a_2\ \ \ a_\ell)$ of elements of A The collection of such sequences is denoted by A^* For s as above, ℓ is the *length* of s (notation $|s|$) We say that s is *empty* if $|s| = 0$

(ii) Let s be a sequence For each $1 \leq i \leq |s|$, the i-th element s is indicated by $(s)_i$

(iii) We use the following denotation for specific parts of non-empty sequences s

$$s^- = (a_2, \quad , a_\ell), \qquad s_- = (a_1, \quad , a_{\ell-1})$$

(iv) Let s, t be sequences Then s and t are *disjoint* (notation $s \# t$) if

$$\forall i, j\ [(s)_i \neq (t)_j]$$

(v) The *concatenation* of s and t is denoted by $s * t$

(vi) Let s, t be sequences Then s is a *prefix of* t (notation $s \subseteq t$) if $s * s' = t$ for some s' Moreover $s \subset t$ denotes that s is a proper prefix of t

Graphs

The objects of our interest are finite directed graphs in which each node has a specific label The number of outgoing edges of a node is determined by its label In the sequel we assume that $\mathcal{N}$ is some basic set of *nodes* (infinite, one usually takes $\mathcal{N} = \mathbb{N}$), and Σ is a (possibly infinite) set of *symbols* with *arity* in $\mathbb{N}$

5 2 2 DEFINITION (i) A *labelled graph* (over $\langle \mathcal{N}, \Sigma \rangle$) is a triple

$$g = \langle N, symb, args \rangle$$

such that

(1) $N \subseteq \mathcal{N}$, N is the set of *nodes* of g,

(2) $symb \quad N \to \Sigma$, $symb(n)$ is the *symbol* at node n,

(3) $args \quad N \to N^*$ such that $|args(n)| = \text{arity}(symb(n))$

Thus $args(n)$ specifies the outgoing edges of n

(ii) Let $n \in g$ and $i \leq \text{arity}(n)$ The combination $a = (n, i)$ is called a *reference* The node n is the *root* of a (notation $\text{r}(a)$) Moreover i is the *index* of a (notation $[a]$) The *destination* of a (notation $\text{d}(a)$) is the node to which it refers, i e $args(n)_i$ The set of all references of g is indicated by Ref_g

(iii) A *rooted graph* is a quadruple

$$g = \langle N, symb, args, r \rangle$$

such that $\langle N, symb, args \rangle$ is a labelled graph, and $r \in N$ The node r is called the *root* of the graph g

(iv) The collection of all finite rooted labelled graphs over $\langle \mathcal{N}, \Sigma \rangle$ is indicated by $\mathbb{G}$

CONVENTION (i) m, n, n', range over nodes, g, g', h, range over (rooted) graphs a, b, a', range over references,

(ii) If g is a (rooted) graph, then its components are referred to as N_g, $symb_g$, $args_g$ (and r_g) respectively To simplify notation we write $n \in g$ instead of $n \in N_g$

Paths

5 2 3 DEFINITION (i) Let $a, a' \in Ref_g$ Then a' *succeeds* a if $\text{d}(a) = \text{r}(a')$

(ii) A *path* in a graph is a sequence p of references such that $(p)_{k+1}$ succeeds $(p)_k$ for all $k < |p|$

(iii) Let $a \in Ref_g$ A path p is *extendible with* a if either p is empty, or a succeeds $(p)_{|p|}$ The *extension* of p with a is denoted by $p * a$

(iv) $p \quad n \rightsquigarrow m$ denotes that p *leads from* n *to* m More formally,

$$() \quad n \rightsquigarrow n,$$

$$p \quad n \rightsquigarrow m \quad \Rightarrow \quad p * (m\ i) \quad n \rightsquigarrow args_g(m)_i$$

(v) Let $m, n \in g$ Then m is *reachable from* n (notation $n \rightsquigarrow m$) if $p \quad n \rightsquigarrow m$ for some path p in g

(vi) Let $n \in g$ We write $n \in p$ if $\text{r}((p)_i) = n$ for some i

5 2 4 DEFINITION Let p be a non empty path

(i) The *root* of p (notation $\mathrm{r}(p)$) is the root of its first reference; the *index* of p is the index of this reference. The *destination* of p (notation $\mathrm{d}(p)$) is the destination of its last reference.

(ii) The *node sequence of* p (notation $\tilde{p}$) is the sequence

$$(\mathrm{r}((p)_1), . \quad , \mathrm{r}((p)_{|p|}), \mathrm{d}((p)_{|p|})).$$

(iii) p is a *cycle* if $\mathrm{r}(p) = \mathrm{d}(p)$.

(iv) p is called *cyclic* if it has a cycle as subpath. Moreover p is *root cyclic* if an initial part of p is a cycle.

5.2.5. DEFINITION. A graph g is a *tree* if for each $n \in g$ there exists a unique path leading from r_g to n. This unique path will be denoted as n. Sometimes the last reference of a non-empty n plays a special role, we denote it by n.

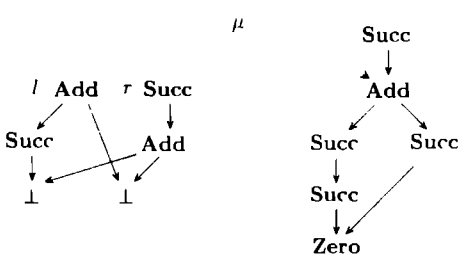
5.2.6. DEFINITION. Let g be a graph and $n \in g$. The *subgraph of* g *at* n (notation $g \mid n$) is the rooted graph $\langle N, symb, args, n\rangle$ where $N = \{m \in g \mid n \rightsquigarrow m\}$, and $symb$ and $args$ are the restrictions (to N) of $symb_g$ and $args_g$ respectively.

Graph rewriting

Rewrite rules specify transformations of graphs. Each rewrite rule is represented by a special graph containing two roots. These roots determine the left-hand side (the *pattern*) and the right-hand side of the rule. Variables are represented by 'empty nodes', containing the special symbol $\perp$. Let R be some rewrite rule. A graph g can be *rewritten* according to R if R is applicable to g, i.e. the pattern of R *matches* g. A *match* is a mapping from the pattern of R to a subgraph of g that preserves the node structure. The following picture indicates a redex, consisting of (the graph representation of) the rule

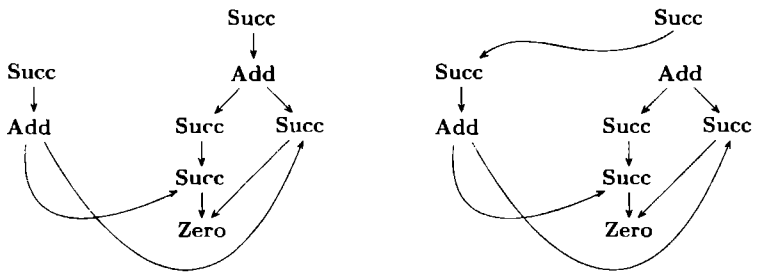
$$\mathbf{Add}(\mathbf{Succ}(x), y) \rightarrow \mathbf{Succ}(\mathbf{Add}(x, y))$$

and a match μ.

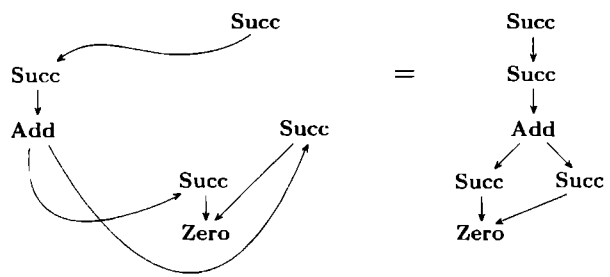


Instead of using names for variables in rewrite rules, multiple occurrence of the same variable is indicated via sharing. E.g. all occurrences of the variable x in the above rule are represented by references to the same empty node.

The combination of a rule and a match is called a *redex*. If a redex has been determined, the graph can be rewritten according to the structure of the right-hand side of the rule involved. This is done in three steps. Firstly, the graph is *extended* with an instance of the right-hand side of the rule. The connections from the new part with the original graph are determined by μ. Then all references to the root of the redex are *redirected* to the root of the right-hand side.



Finally all unreachable nodes are removed by performing *garbage collection*.



This procedure is formalized in the rest of the present section

5.2.7. DEFINITION Let g be a graph.

(i) The set of *empty nodes* of g (notation g°) is the collection

$$g^\circ = \{n \in g \mid symb_g(n) = \bot\}.$$

(ii) The set of *non-empty nodes* (or *interior*) of g is denoted by $g^\bullet$. So $N_g = g^\circ \cup g^\bullet$.

(iii) g is *closed* if $g^\circ = \emptyset$.

The objects on which computations are performed are closed graphs; the others are used as auxiliary objects. e.g. for defining graph rewrite rules.

5.2.8. DEFINITION. (i) A *term graph rewrite rule* (or *rule* for short) is a triple $R = \langle g, l, r\rangle$ where g is a (possibly open) graph, and $l, r \in g$ (called the *left root* and *right root* of R), such that

(1) $(g \mid l)^\bullet \neq \emptyset$;

(2) $(g \mid r)^\circ \subseteq (g \mid l)^\circ$.

(ii) If $symb_g(l) = F$ then R is said to be a *rule for* F.

(iii) A symbol S is a *pattern symbol* of R if $S = symb_g(n)$ for some $n \in R|l, n \neq l$.

(iv) R is *left-linear* if $g \mid l$ is a tree

(v) R is *extending* if $r \notin (g \mid l)$. Otherwise, R is *projecting*.

In (i) above, condition (1) expresses that the left-hand side of the rewrite rule should not be just a variable. Moreover condition (2) states that all variables occurring on the right-hand side of the rule should also occur on the left-hand side.

NOTATION. We will write $R \mid l$, $R \mid r$ for $g_R \mid l_R$, $g_R \mid r_R$ respectively.

5.2.9. DEFINITION. Let p, g be graphs.

(i) A *match* is a function $\mu : N_p \to N_g$ such that for all $n \in p^\bullet$

$$\begin{aligned} symb_g(\mu(n)) &= symb_p(n), \\ args_g(\mu(n))_i &= \mu(args_p(n)_i). \end{aligned}$$

In this case we write $\mu : p \underset{m}{\rightarrow} g$.

(ii) μ is a *rooted match* from p to g (notation $\mu : p \underset{rm}{\rightarrow} g$) if $\mu : p \underset{m}{\rightarrow} g$ and $\mu(r_p) = \mu(r_g)$.

(iii) By $p \underset{m}{\rightarrow} g$ we will denote that there exists a match $\mu : p \underset{m}{\rightarrow} g$. Analogously we write $p \underset{rm}{\rightarrow} g$ for rooted matches.

(iv) The graphs g and g' are *compatible* (notation $g \uparrow g'$) if for some h one has $g \underset{rm}{\rightarrow} h$ and $g' \underset{rm}{\rightarrow} h$.

We now formalize the rewrite procedure.

5.2.10 DEFINITION. Let g be a graph, and $\mathcal{R}$ a set of rewrite rules. An *$\mathcal{R}$-redex* in g (or just *redex*) is a tuple $\Delta = \langle R, \mu \rangle$ where $R \in \mathcal{R}$, and $\mu : (R \mid l) \underset{m}{\rightarrow} g$.

5.2.11. DEFINITION. Let N, A be sets of nodes. The *disjoint set extension* of N with A (notation $N \cup^+ A$) is the set $N \cup A^*$, where $A^* = \{a^* \mid a \in A\}$ is a set of fresh nodes associated with A.

5.2.12. DEFINITION. Let g be a graph. Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Set $N = (R \mid r)^\bullet \backslash (R \mid l)^\bullet$ (the set of new nodes).

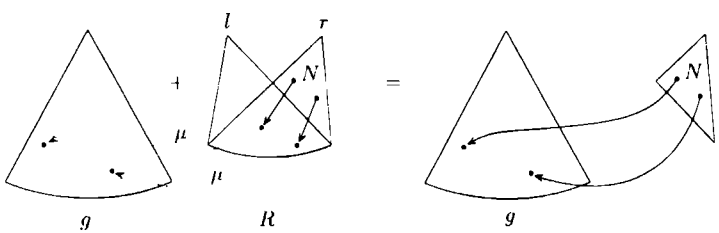
(i) The Δ-*extension* of g (notation $g + \Delta$) is defined as follows. Then $g + \Delta = h$, where

$$\begin{aligned}
N_h &= N_g \cup^+ N; \\
symb_h(n) &= symb_g(n) && \text{if } n \in g, \\
&= symb_R(n) && \text{otherwise;} \\
args_h(n)_i &= args_g(n)_i && \text{if } n \in g, \\
&= args_R(n)_i && \text{if } n, args_R(n) \in N, \\
&= \mu(args_R(n)_i) && \text{otherwise;} \\
r_h &= r_g.
\end{aligned}$$

(ii) The *contractum root* of Δ (notation $\mathrm{r}(\Delta)$) is defined as follows.

$$\begin{aligned}
\mathrm{r}(\Delta) &= r_R && \text{if } r_R \in N, \\
&= \mu(r_R) && \text{otherwise.}
\end{aligned}$$

This construction can be visualized as follows.



A redirection in a graph is an operation which replaces all references to a given node by references to another one.

5.2.13. DEFINITION. Let g be a graph. Let $n, m \in g$.

(i) The *redirection function* associated with n, m (notation $[n \mapsto m]$) on N_g maps n to m and is the identity elsewhere.

(ii) The result of *redirecting* n to m (notation $g[n := m]$) is the graph

$$\langle N_g, symb_g, args', [n \mapsto m](r_g) \rangle$$

where $args'(x)_i = [n \mapsto m](args_g(x)_i)$.

A rewrite step is concluded by garbage collection.

5.2.14. DEFINITION. Let g be a rooted graph. The result of performing *garbage collection* on g (notation $\mathrm{GC}(g)$) is the graph obtained from g by deleting all nodes that are not reachable from its root, i.e. $\mathrm{GC}(g) = g \mid r_g$.

The three basic operations extension, redirection, and garbage collection are combined into a single operation This mechanism corresponds to *substitution* in lambda calculus and term rewriting

5 2 15 DEFINITION Let Δ be a ($\mathcal{R}$)redex in g The result h of *contracting* Δ in g is defined by

$$h = \mathrm{GC}((g + \Delta)[\mu(l) = \mathrm{r}(\Delta)])$$

In this case we write $g \xrightarrow[\mathcal{R}]{\Delta} h$, or just $g \to h$ We call h the Δ-*reduct* of g

Term graph rewrite systems

A collection of graphs and a set of rewrite rules can be combined into a (term) graph rewrite system A special class of so-called orthogonal graph rewrite systems is the subject of further investigations

5 2 16 DEFINITION (i) A *term graph rewrite system* (TGRS) is a tuple $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{R}$ is a set of rewrite rules, and $\mathcal{G} \subseteq \mathbb{G}$ is a set of closed graphs which is closed under $\mathcal{R}$-reduction

(ii) $\mathfrak{T}$ is *left-linear* if each $R \in \mathcal{R}$ is left-linear

(iii) $\mathfrak{T}$ is *regular* if for each $g \in \mathcal{G}$ the $\mathcal{R}$-redexes in g are pairwise disjoint

(iv) $\mathfrak{T}$ is *orthogonal* if $\mathfrak{T}$ is both left-linear and regular

It can be shown that for a large class of orthogonal TGRS's (the so-called *interference-free* systems) the Church-Rosser property holds (see Barendsen and Smetsers (1992))

Applicative graph rewrite systems

In TGRS's, all symbols have a fixed arity This makes it impossible to use *functions* as arguments or functions as result The concept of higher order functions, however, can be simulated as follows

Say we want to model the function $g(x) = \lambda y\ f(x, y)$ The idea is to specify f by a TGRS rule

$$\mathbf{F}(x, y) \to$$

and to associate with $\mathbf{F}$ a so called *Curry variant* $\mathbf{F}_1$ of arity 1 Here $\mathbf{F}_1$ is regarded as a constructor symbol The intention is that an application $\mathbf{F}_1(X)$ stands for $\lambda y\ \mathbf{F}(X, y)$ Moreover one specifies an *application rule* for the special function symbol $\mathbf{Ap}$ (of arity 2) stating

$$\mathbf{Ap}(\mathbf{F}_1(x), y) \to \mathbf{F}(x, y)$$

which supplies the 'partial application' of $\mathbf{F}$ with an extra argument

In general, one can associate with each function symbol F of non-zero arity (say k) a collection of k Curry variants of arity $0, 1, \ldots, k-1$ respectively. The rules for **Ap** look like

$$\begin{aligned} \mathbf{Ap}(F_0, x_1) &\to F_1(x_1), \\ \mathbf{Ap}(F_1(x_1), x_2) &\to F_2(x_1, x_2), \end{aligned}$$

$$\mathbf{Ap}(F_{k-1}(x_1, \ldots, x_{k-1}), x_k) \to F(x_1, \ldots, x_k).$$

The Curry rule for arity j is referred to as Ap_j^F.

In our approach, the basis of a TGRS is formed by proper functional rewrite rules (like the one for **F** above). In the rewrite rules, one might use **Ap** and Curry variants, however. The auxiliary **Ap** rules have the standard shapes as indicated above. We consider *Curry closed* systems, i.e. systems in which there is an **Ap** rule for each occurring Curry variant.

A special class of TGRS's is formed by so-called *Curry complete* TGRS's, in which *all* Curry variants and corresponding application rules are present.

5.2.17. EXAMPLE. *Combinatory Logic* (CL), originally expressed by

$$\begin{aligned} \mathbf{S}xyz &= xz(yz) \\ \mathbf{K}xy &= x \\ \mathbf{I}x &= x \end{aligned}$$

can be formulated as a Curry complete TGRS in the following way.

$$\begin{aligned} \mathbf{S}(x,y,z) &\to \mathbf{Ap}(\mathbf{Ap}(x,z), \mathbf{Ap}(y,z)) \\ \mathbf{K}(x,y) &\to x \\ \mathbf{I}(x) &\to x \\ \mathbf{Ap}(\mathbf{S}_0, x) &\to \mathbf{S}_1(x) \\ \mathbf{Ap}(\mathbf{S}_1(x), y) &\to \mathbf{S}_2(x,y) \\ \mathbf{Ap}(\mathbf{S}_2(x,y), z) &\to \mathbf{S}(x,y,z) \\ \mathbf{Ap}(\mathbf{K}_0, x) &\to \mathbf{K}_1(x) \\ \mathbf{Ap}(\mathbf{K}_1(x), y) &\to \mathbf{K}(x,y) \\ \mathbf{Ap}(\mathbf{I}_0, x) &\to \mathbf{I}(x) \end{aligned}$$

Any Curry complete system is equivalent with a *purely applicative* TGRS (where **Ap** is the only function symbol). For the analysis of typing, however, it will be convenient to focus on the 'functional basis' of a TGRS and view the **Ap** rules as special rules with an induced typing.

5.2.18. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS.

(i) The set of $\mathfrak{T}$-*symbols* (notation $\Sigma_{\mathfrak{T}}$) consists of the symbols appearing in $\mathcal{G}$ or in $\mathcal{R}$.

(ii) $\mathcal{R}$ is divided into two parts: $\mathcal{R}_{\mathcal{F}}$ (proper functional rules) and $\mathcal{R}_{\mathcal{C}}$ (Curry rules for **Ap**).

(iii) The set of *proper function symbols* of $\mathfrak{T}$ (notation $\Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T})$) consists of the symbols having a rule in $\mathcal{R}_{\mathcal{F}}$. Moreover $\Sigma^{\mathrm{c}}_{\mathcal{F}}(\mathfrak{T})$ denotes the set of their Curry variants. These sets combine into the collection $\Sigma_{\mathcal{F}}(\mathfrak{T})$ of *functional symbols* of $\mathfrak{T}$, i.e. $\Sigma_{\mathcal{F}}(\mathfrak{T}) = \Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T}) \cup \Sigma^{\mathrm{c}}_{\mathcal{F}}(\mathfrak{T})$.

(iv) The set of *function symbols* of $\mathfrak{T}$ (notation $\mathrm{Fun}(\mathfrak{T})$) consists of those symbols having a rule in $\mathfrak{T}$. In general one has $\mathrm{Fun}(\mathfrak{T}) = \Sigma^{\mathrm{f}}_{\mathcal{F}} \cup \{\mathbf{Ap}\}$. The other symbols are called *data symbols* (notation $\mathrm{Data}(\mathfrak{T})$). Note that Curry variants are data symbols.

We will only consider *function-data systems*, i.e. systems in which all pattern symbol are included in $\mathrm{Data}(\mathfrak{T})$. We will usually omit '$(\mathfrak{T})$' in the denotations introduced above if $\mathfrak{T}$ is fixed.

5.3. Conventional Typing

In this section we will define a notion of simple type assignment to graphs using a type system based on traditional systems for functional languages. The approach is similar to the one introduced in van Bakel et al. (1992). It is meant to illustrate the concept of 'classical' typing for graphs.

5.3.1 DEFINITION. Let $\mathbb{V}$ be a set of *type variables*, and $\mathbb{C}$ a set of *type constructors* with *arity* in $\mathbb{N}$.

(i) The set $\mathbb{T}$ of (*graph*) *types over* $\mathbb{C}$ is defined inductively as follows.

$$\begin{aligned} \alpha \in \mathbb{V} &\Rightarrow \alpha \in \mathbb{T}, \\ \sigma, \tau \in \mathbb{T} &\Rightarrow \sigma \to \tau \in \mathbb{T}, \\ T \in \mathbb{C}, \sigma_1, \ldots, \sigma_k \in \mathbb{T} &\Rightarrow T(\sigma_1, \ldots, \sigma_k) \in \mathbb{T}. \end{aligned}$$

(ii) $\mathrm{TV}(\sigma)$ denotes the set of type variables occurring in σ. Let $\vec{\alpha} \in \mathbb{V}$. Then $\mathbb{T}(\vec{\alpha})$ denotes the set of types σ with $\mathrm{TV}(\sigma) \subseteq \{\vec{\alpha}\}$.

(iii) The set $\mathbb{T}_{\mathrm{S}}$ of *symbol types* is defined as

$$\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{T} \Rightarrow (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau \in \mathbb{T}_{\mathrm{S}}, \qquad k \geq 0$$

We will usually abbreviate $() \rightarrowtail \tau$ to τ and $(\sigma) \rightarrowtail \tau$ to $\sigma \rightarrowtail \tau$.

CONVENTION. In the sequel, $\alpha, \beta, \alpha_1, \ldots$ range over type variables; $\sigma, \tau, \tau_1, \ldots$ range over (symbol) types.

5.3.2. DEFINITION. (i) A *substitution* is a function $* : \mathbb{V} \to \mathbb{T}$. This induces an operation on $\mathbb{T}$ (and $\mathbb{T}_{\mathrm{S}}$) in a straightforward way.

(ii) σ is an *instance* of τ (notation $\sigma \subseteq \tau$) if there exists a substitution $*$ such that $\tau^* = \sigma$.

(iii) Let $\sigma, \tau \in \mathbb{T}$. A *unifier* for σ and τ is a substitution $*$ such that $\sigma^* = \tau^*$.

(iv) The above substitution $*$ is a *most general unifier* if for all $*_1$

$$\sigma^{*_1} = \tau^{*_1} \Rightarrow *_1 = *_2 \circ * \text{ for some } *_2 .$$

Algebraic type systems

In our system we consider types which are built up from type variables and type constructors. The standard type constructor is $\rightarrow$ for function spaces. We will use the infix denotation $\sigma \rightarrow \tau$ for function types.

The other type constructors are assumed to be given by a so called *algebraic type system*. The idea is to specify the *canonical* inhabitants of each constructor type $T\vec{\sigma}$ by a declaration of the form

$$T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$$

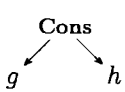
where the length of $\vec{\alpha}$ indicates the arity of the defined type constructor T. A subexpression $C_i\vec{\sigma_i}$ is called a *clause* in the definition of $T\vec{\alpha}$. The free variables in σ_i are among the $\vec{\alpha}$. Moreover C_i is called an *algebraic constructor*.

An *algebraic type system* is a collection $\mathcal{A}$ of constructor declarations. These might be (directly or indirectly) recursive.

5.3.3. EXAMPLE. The system

$$\begin{aligned} \mathrm{Nat} &= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \\ \mathrm{List}(\alpha) &= \mathbf{Cons}(\alpha, \mathrm{List}(\alpha)) \mid \mathbf{Nil} \end{aligned}$$

specifies that the (canonical) objects of type Nat are the constants **0**, **1**, **2**, etc. Moreover an object of type List(σ) is either equal to **Nil** or to a graph of the form



where g is an object of type σ, and h is again an object of type List(σ)

If $\mathcal{A}$ is an algebraic type system, then $\mathbb{C}_{\mathcal{A}}$ denotes the set of type constructors specified in $\mathcal{A}$. Moreover $\Sigma_{\mathcal{A}}$ is the set of (graph) constructors introduced in $\mathcal{A}$. In the above example one has

$$\mathbb{C}_{\mathcal{A}} = \{\mathrm{Nat}, \mathrm{List}\}$$

and

$$\Sigma_{\mathcal{A}} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{Cons}, \mathbf{Nil}\}.$$

It is assumed that each algebraic constructor is used only once in $\mathcal{A}$

We restrict ourselves to the following combinations of algebraic type systems and TGRS's We write Σ_0 for the set of *basic symbols* $\bot$ and **Ap**

5 3 4 DEFINITION Let $\mathcal{A}$ be an algebraic type system, and let $\mathfrak{T}$ be a TGRS Then $\mathfrak{T}$ is *applicative over* $\mathcal{A}$ if $\mathfrak{T}$ is Curry closed, and $\mathrm{Data}(\mathfrak{T}) \subseteq \Sigma_0 \cup \Sigma^c_{\mathcal{F}}(\mathfrak{T}) \cup \Sigma_{\mathcal{A}}$

We restrict our types accordingly by setting $\mathbb{C} = \mathbb{C}_{\mathcal{A}}$

Type assignment for graphs

In the rest of this section we describe how types can be assigned to graphs given a type assignment to the symbols by a so called *environment*

5 3 5 DEFINITION (i) Let $\Gamma \subseteq \Sigma$ be a set of symbols A *type environment for* Γ is a function $\mathcal{E}$ $\Gamma \to \mathbb{T}$

(ii) The *basic type environment* $\mathcal{E}_0$ is the following environment for Σ_0

$$\begin{aligned} \mathcal{E}_0(\bot) &= \alpha, \\ \mathcal{E}_0(\mathbf{Ap}) &= (\alpha \to \beta, \alpha) \rightarrowtail \beta \end{aligned}$$

These environment types are regarded as *type schemes*, i e in any concrete application of a symbol one uses an instance of its environment type An alternative view is to consider the environment types as universally quantified on the topmost level, e g $\forall\alpha\forall\beta\ (\alpha \to \beta, \alpha) \rightarrowtail \beta$

5 3 6 DEFINITION Let $g = \langle N, symb, args\rangle$ be a graph

(i) A *type assignment* to g (or *typing for* g) is a function $\mathcal{T}$ $N \to \mathbb{T}$

(ii) Let $\mathcal{T}$ be a typing for g, and $n \in g$, say $k = \mathrm{arity}(n)$ The *symbol type* of n according to $\mathcal{T}$ (notation $\mathcal{F}_{\mathcal{T}}(n)$) is defined by

$$\mathcal{F}_{\mathcal{T}}(n) = (\mathcal{T}(args(n)_1), \quad , \mathcal{T}(args(n)_k)) \rightarrowtail \mathcal{T}(n)$$

(iii) Let $\mathcal{E}$ be an environment Then $\mathcal{T}$ is an *$\mathcal{E}$-typing for* g if for each $n \in g$ one has

$$\mathcal{F}_{\mathcal{T}}(n) \subseteq \mathcal{E}(symb(n))$$

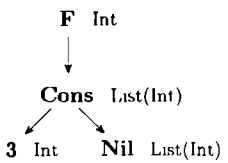
(iv) g is *$\mathcal{E}$-typable with type* σ (notation $\mathcal{E} \vdash g$ σ) if there exists an $\mathcal{E}$-typing $\mathcal{T}$ for g with $\mathcal{T}(r_g) = \sigma$

(v) If $\mathcal{E} \vdash g$ σ is a typing statement, then g is called the *subject* of the statement, moreover σ is its *predicate*

5.3.7. EXAMPLE. Let $\mathcal{E}$ be an environment containing the type declarations

F	:	$\mathrm{List}(\beta) \rightarrowtail \beta$,
Cons	:	$(\alpha, \mathrm{List}(\alpha)) \rightarrowtail \mathrm{List}(\alpha)$,
Nil	:	$\mathrm{List}(\alpha)$,
3	:	Int

The following type assignment shows that $\mathcal{E} \vdash g : \mathrm{Int}$ for the the displayed graph g.



Type assignments are substitutive. This is formulated as follows.

5.3.8. INSTANTIATION LEMMA. *Let g be a graph, and $\mathcal{E}$ an environment. Let $*$ be a substitution.*

(i) *Let $\mathcal{T}$ be an $\mathcal{E}$-typing for g. Then $\mathcal{T}^*$ is an $\mathcal{E}$-typing for g.*

(ii) *For each $\sigma \in \mathbb{T}$*

$$\mathcal{E} \vdash g \cdot \sigma \;\Rightarrow\; \mathcal{E} \vdash g : \sigma^*.$$

PROOF. Easy. □

Principal types

In this subsection we will show that type assignment has the principal type property, i.e. if a graph g is typable then there exists a most general typing for g. In the sequel, some fixed environment $\mathcal{E}$ is assumed. The presentation below is inspired by Barendregt (1992). The idea of splitting type reconstruction into generation of equations and unification is due to Wand (1987).

5.3.9 DEFINITION. Let $E = \{\sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n\}$ be a finite set of equations between types. A *solution* for E is a substitution $*$ such that

$$\sigma_1^* = \tau_1^* \;\&\; \cdots \;\&\; \sigma_n^* = \tau_n^*.$$

In that case one writes $* \models E$. Analogously to the notion of most general unifier, one defines the notion of *most general solution* for E.

Note that a most general solution is unique up to renaming of type variables.

5.3.10. UNIFICATION THEOREM. *There exists a recursive function* Unify *having as input finite sets of equations between types such that*

$$\begin{aligned} E \text{ has a solution} &\Rightarrow \text{Unify}(E) \text{ is a most general solution for } E; \\ E \text{ has no solution} &\Rightarrow \text{Unify}(E) = \text{fail}. \end{aligned}$$

PROOF. See Robinson (1965). □

5.3.11. PROPOSITION. *Let g be a graph, and $\mathcal{T}$ a typing for g. Then there exist a finite set of equations $E = E_{\mathcal{E}}(\mathcal{T}, g)$ such that for all substitutions $*$ one has*

(1) $* \models E \quad \Rightarrow \quad \mathcal{T}^*$ *is an $\mathcal{E}$-typing for g.*

(2) $\mathcal{T}^*$ *is a typing for g* $\quad \Rightarrow \quad *_1 \models E$ *for some $*_1$ such that $\mathcal{T}^{*_1} = \mathcal{T}^*$.*

PROOF. Define

$$\begin{aligned} E_{\mathcal{E}}(\mathcal{T}, n) &= [\, \mathcal{F}_{\mathcal{T}}(n) = \mathcal{E}(symb(n)) \,] \\ E_{\mathcal{E}}(\mathcal{T}, g) &= \{E_{\mathcal{E}}(\mathcal{T}, n) \mid n \in g\}. \end{aligned}$$

We assume that in each equation of $E_{\mathcal{E}}(\mathcal{T}, g)$ a 'fresh' instance of $\mathcal{E}(symb(n))$ is chosen, i.e an instance having no variables in common with any other type appearing in $E_{\mathcal{E}}(\mathcal{T}, g)$.

(1) Obvious.

(2) Since $\mathcal{T}^*$ is an $\mathcal{E}$-typing for g, for each $n \in g$ there exists a substitution $*_n$ such that

$$(\mathcal{F}_{\mathcal{T}}(n))^* = \mathcal{E}(symb(n))^{*_n}$$

Observe that each substitution $*_n$ has only on effect on the corresponding instance of $\mathcal{E}(symb(n))$. Hence one can write $*_1$ as a composition of $*$ and all substitutions $*_n$ for $n \in g$. □

5.3.12. DEFINITION. Let g be a graph, and $\mathcal{T}$ an $\mathcal{E}$-typing for g. $\mathcal{T}$ is a *principal* $\mathcal{E}$-typing for g if

$$\mathcal{T}' \text{ is an } \mathcal{E}\text{-typing for } g \;\Rightarrow\; \mathcal{T}' = \mathcal{T}^* \text{ for some } *.$$

5.3.13. PRINCIPAL TYPE THEOREM. *There exists a recursive function pt such that*

$$\begin{aligned} g \text{ is } \mathcal{E}\text{-typable} &\Rightarrow pt(g) \text{ is a principal } \mathcal{E}\text{-typing for } g; \\ g \text{ is not } \mathcal{E}\text{-typable} &\Rightarrow pt(g) = \text{fail}. \end{aligned}$$

PROOF. Set $\mathcal{T}_0(n) = \alpha_n$ where α_n is fresh. Define

$$\begin{aligned} pt(g) &= \mathcal{T}_0^* \qquad \text{if Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = *, \\ &= \text{fail} \qquad \text{if Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = \text{fail} \end{aligned}$$

Suppose g is $\mathcal{E}$-typable, and $\mathcal{T}$ is an $\mathcal{E}$-typing for g. Obviously, $\mathcal{T} = \mathcal{T}_0^{*_0}$ for some $*_0$. By Proposition 5.3.11 (2), there exist an $*_1$ such that $*_1 \models E_\mathcal{E}(\mathcal{T}_0, g)$ and $\mathcal{T}_0^{*_1} = \mathcal{T}_0^{*_0}$. Consequently, $\mathrm{Unify}(E_\mathcal{E}(\mathcal{T}_0, g)) = *$ is defined and $\mathcal{T}_0^*$ is a typing for g. Since $*$ is a most general unifier of $E_\mathcal{E}(\mathcal{T}_0, g)$ it follows that $*_1 = *_2 \circ *$ for some $*_2$. Now

$$(\mathcal{T}_0^*)^{*_2} = \mathcal{T}_0^{*_1} = \mathcal{T}_0^{*_0} = \mathcal{T}.$$

If g is not typable, then there exists no $*$ such that $* \models E_\mathcal{E}(\mathcal{T}_0, g)$, and hence

$$\mathrm{Unify}(E_\mathcal{E}(\mathcal{T}_0, g)) = \mathsf{fail} = pt(g). \ \square$$

Type assignment for rewrite rules

5.3.14 DEFINITION. (i) Let $g = \langle h, l, r\rangle$ be a graph with two roots. A *balanced $\mathcal{E}$-typing* for g is an $\mathcal{E}$-typing $\mathcal{T}$ for h such that

$$\mathcal{T}(l) = \mathcal{T}(r).$$

(ii) The denotation $\mathcal{E} \vdash R : \vec{\sigma} \rightarrowtail \tau$ indicates that there exists a balanced $\mathcal{E}$-typing $\mathcal{T}$ for g_R with $\mathcal{F}_\mathcal{T}(l) = \vec{\sigma} \rightarrowtail \tau$.

5.3.15. DEFINITION. (i) Let R be a rewrite rule. An *$\mathcal{E}$-typing* for R is a type assignment $\mathcal{T}$ to g that meets the following requirements.

(1) $\mathcal{T}$ is a balanced $\mathcal{E}$-typing $\mathcal{T}$ for R;

(2) $\mathcal{T} \restriction (R \mid l) = pt(R \mid l)$.

(ii) A set $\mathcal{R}$ of rewrite rules is *$\mathcal{E}$-typable* if each $R \in \mathcal{R}$ has an $\mathcal{E}$-typing.

The requirement for mentioned in (2) ensures that the actual typing of a graph (restricted to the matching part) is an instance of the typing of R (restricted to $R \mid l$) whenever R is applicable.

Type environments

In this subsection we will describe two standard constructions for type environments: an environment $\mathcal{E}_\mathcal{A}$ based on the algebraic type system $\mathcal{A}$, and an environment $\mathcal{E}_\mathcal{F}$ for functional symbols, based on a type assignment to the proper function symbols.

5.3.16. DEFINITION. Let $\mathcal{A}$ be an algebraic type system. The *algebraic environment* for $\Sigma_\mathcal{A}$ associated with $\mathcal{A}$ (notation $\mathcal{E}_\mathcal{A}$) is obtained by setting for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$ and each i

$$\mathcal{E}_\mathcal{A}(C_i) = \vec{\sigma_i} \rightarrowtail T\vec{\alpha}.$$

5.3.17. DEFINITION. For each symbol type $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ and $j \leq k$, the *j-th Curried version* of σ (notation σ^C_j) is the type

$$(\sigma_1, \ldots, \sigma_j) \rightarrowtail \sigma_{j+1} \rightarrow (\sigma_{j+2} \rightarrow (\cdots \rightarrow (\sigma_k \rightarrow \tau)\cdots)).$$

5.3.18. DEFINITION. (i) A *function type environment* is a map $\mathcal{F} : \Sigma^{\mathrm{f}}_{\mathcal{F}} \rightarrow \mathbb{T}_{\mathrm{S}}$.

(ii) Such a function type environment induces a type environment for $\Sigma_{\mathcal{F}}$ (notation $\mathcal{E}_{\mathcal{F}}$) in the following way.

$$\begin{aligned} \mathcal{E}_{\mathcal{F}}(F) &= \mathcal{F}(F), \\ \mathcal{E}_{\mathcal{F}}(F_j) &= \mathcal{F}(F)^C_j. \end{aligned}$$

These constructions are combined in the following way.

5.3.19. DEFINITION. Let $\mathfrak{T}$ be an applicative TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment.

(i) The *combined type environment* $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is obtained by setting

$$\mathcal{E}_{\mathcal{F},\mathcal{A}} = \mathcal{E}_0 \cup \mathcal{E}_{\mathcal{F}} \cup \mathcal{E}_{\mathcal{A}}.$$

(ii) Typability in $\mathcal{F}, \mathcal{A}$ is denoted by writing $\mathcal{F}, \mathcal{A} \vdash g : \sigma$ instead of $\mathcal{E}_{\mathcal{F},\mathcal{A}} \vdash g : \sigma$.

Subject reduction

In this subsection we will show that typing is preserved under rewriting.

First, we will derive some properties concerning the interaction between typing and the basic graph operations matching, extension, redirection and garbage collection. For the moment, fix some type environment $\mathcal{E}$.

5.3.20. MATCHING LEMMA. *Let p, g be graphs, and let $\mu : p \xrightarrow[m]{} g$ be a match. Furthermore, let $\mathcal{T}$ be a typing for g. Then $\mathcal{T} \circ \mu$ is a typing for p.*

PROOF. Let $n \in p$. The case $n \in p^\circ$ is trivial. Suppose $n \in p^\bullet$. Then

$$\mathcal{F}_{\mathcal{T}\circ\mu}(n) \subseteq \mathcal{E}(symb(\mu(n))) = \mathcal{E}(symb(n)). \ \square$$

5.3.21. EXTENSION LEMMA. *Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Let $\mathcal{T}_g, \mathcal{T}$ be typings for g and $R \mid r$ respectively. Set $O = (R \mid r) \cap (R \mid l)$. Then*

$$\mathcal{T} \restriction O = (\mathcal{T}_g \circ \mu) \restriction O \ \Rightarrow \ \mathcal{T}_g + \mathcal{T} \text{ is a typing for } g + \Delta.$$

Here $\mathcal{T}_g + \mathcal{T}$ denotes the extension of $\mathcal{T}_g$ with $\mathcal{T}$

PROOF. By a case distinction. The case $n \in g$ is trivial. Suppose $n \in (R \mid r)^\bullet \backslash (R \mid l)^\bullet$. Observe that $\mathcal{T}_g + \mathcal{T}(args_{g+\Delta}(n)_i) = \mathcal{T}(args_R(n)_i)$ for any i. Hence $\mathcal{F}_{\mathcal{T}_g+\mathcal{T}}(n) \subseteq \mathcal{E}(symb(n))$. $\square$

5.3.22. REDIRECTION LEMMA. *Let $\mathcal{T}$ be a typing for g. Furthermore, let $m, n \in g$. If $\mathcal{T}(m) = \mathcal{T}(n)$ then $\mathcal{T}$ is a typing for $g[m := n]$.*

PROOF. Trivial. □

5.3.23. GARBAGE COLLECTION LEMMA. *Let $\mathcal{T}$ be a typing for g. Then $\mathcal{T} \restriction N_{\mathrm{GC}(g)}$ is a typing for $\mathrm{GC}(g)$.*

PROOF. Trivial. □

5.3.24 SUBJECT REDUCTION THEOREM. *Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$. Suppose $\mathcal{R}$ is $\mathcal{E}$-typable. Then for any $g, h \in \mathcal{G}$*

$$\mathcal{E} \vdash g : \sigma,\ g \to h \ \Rightarrow\ \mathcal{E} \vdash h : \sigma.$$

PROOF. Let $\mathcal{T}$ be an $\mathcal{E}$-typing for g. Let $\mathcal{T}_R$ be an $\mathcal{E}$-typing for R. Consider $\mathcal{T} \circ \mu$. By the Matching Lemma this is a typing for $R \mid l$. Hence $\mathcal{T} \circ \mu = (\mathcal{T}_R \restriction (R \mid l))^*$ for some $*$, by the principal type property. Then $\mathcal{T}_R^*$ is a typing for $R \mid r$ by the Instantiation Lemma 5.3.8. Set $\mathcal{T}_h = \mathcal{T} + \mathcal{T}_R^*$. This is a typing for $g + \Delta$, by the Extension Lemma. Note that $\mathcal{T}_h(r(\Delta)) = \mathcal{T}(\mu(l))$. Now we are done by the Redirection Lemma and the Garbage Collection Lemma. □

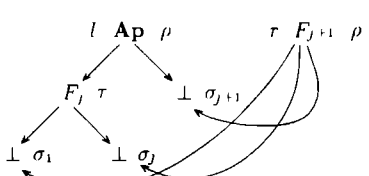
We now consider typings for applicative TGRS's. Until now, rules in $\mathcal{R}_{\mathcal{F}}$ and $\mathcal{R}_{\mathcal{C}}$ have been treated in the same way. It is intuitively clear that the typing for proper function symbols and their Curry variants are related. For applicative systems we will consider environments with the following property.

5.3.25. DEFINITION. Let $\mathcal{E}$ be a type environment for $\mathfrak{T}$. Then $\mathcal{E}$ is said to be *applicative* if for any $F \in \Sigma_{\mathcal{F}}^{\mathrm{f}}$ and $j < \mathrm{arity}(F)$

$$\mathcal{E}(F_j) = \mathcal{E}(F)_j^{\mathcal{C}}.$$

In such an applicative environment the typing notion for $\mathcal{R}_{\mathcal{C}}$ is straightforward.

5.3.26. DEFINITION. Let $F \in \Sigma_{\mathcal{F}}$ with arity $k \geq 1$. Say $\mathcal{F}(F) = (\sigma_1, \ldots, \sigma_k) \mapsto \sigma_{k+1}$. Let $j < k$. Set $\rho = \sigma_{j+2} \to \cdots \to \sigma_{k+1}$, and $\tau = \sigma_{j+1} \to \rho$. The *standard typing* for Ap_j^F is the type assignment indicated by the following picture. (For simplicity we write F as F_k.)



5.3.27. LEMMA. *Typing in applicative environments satisfies the subject reduction property for Curry reductions.*

PROOF. Obvious since the above typing is a rule typing for each Ap_j^F. □

In the remainder of this subsection we will formulate a criterion for balanced typings of functional rewrite rules which implies the subject reduction property for the complete applicative TGRS. It is possible to express a sufficient typing condition avoiding the principal type property. The results apply to a large class of TGRS's and applicative type environments, notably those related to functional programming languages.

5.3.28. DEFINITION. Let $\mathcal{E}$ be an environment for $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$. Then $\mathcal{E}$ is called *principal* for $\mathfrak{T}$ if for each $R \in \mathcal{R}_{\mathcal{F}}$ (say for F) one has

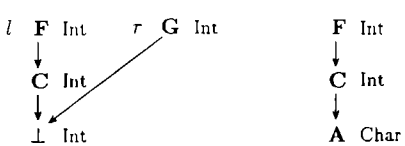
$$\mathcal{E} \vdash R \,.\, \mathcal{E}(F).$$

Requiring that $\mathcal{E}$ is principal is not sufficient to guarantee preservance of typing under reduction.

5.3.29. NON-EXAMPLE. Let $\mathcal{E}$ contain the following type declarations.

F	:	$\mathrm{Int} \rightarrowtail \mathrm{Int}$,
C	:	$\alpha \rightarrowtail \mathrm{Int}$,
G	:	$\mathrm{Int} \rightarrowtail \mathrm{Int}$,
A	:	Char.

The respective typings of the rule $\mathbf{F}(\mathbf{C}(x)) \rightarrow \mathbf{G}(x)$ and the graph $g = \mathbf{F}(\mathbf{C}(\mathbf{A}))$ are



However, rewriting g results in the graph $\mathbf{G}(\mathbf{A})$ which is not typable.

The problem arises from the fact that the typing of the matching part of the object graph cannot be extended to a typing of the right-hand side of the rule. In the example, observe that typing the right-hand side of the rule for **F** results in a type for $\perp$ (namely Int) that is not induced by the environment type for **F**. This leaves some freedom, admitting type-correct object graphs conflicting the assumed left-hand side typing in lower pattern nodes. The idea is to force encoding of pattern types in the environment type for **F**; thus the type structure of any matching part is uniquely determined by this environment type. This is established by a requirement on environment types of pattern symbols

5 3 30 DEFINITION (i) A symbol type $(\sigma_1, \quad, \sigma_k) \rightarrowtail \tau$ is *argument preserving* if

$$\forall i \leq k \, [\, \mathrm{TV}(\sigma_i) \subseteq \mathrm{TV}(\tau) \,]$$

(ii) Let $\mathcal{E}$ be an environment, and $S \in \Sigma$ Then $\mathcal{E}$ is *argument preserving* w r t S if $\mathcal{E}(S)$ is argument preserving

(iii) Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS Then $\mathcal{E}$ is *safe* for $\mathfrak{T}$ if $\mathcal{E}$ is principal for $\mathfrak{T}$ and $\mathcal{E}$ is argument preserving w r t each pattern symbol of $\mathcal{R}_{\mathcal{F}}$

In order to prove that any $\mathcal{E}$-typing for $\mathfrak{T}$ satisfies the subject reduction prop erty whenever $\mathcal{E}$ safe for $\mathfrak{T}$, it will be sufficient to show that the type variables occurring in a pattern typing are propagated upwards

5 3 31 VARIABLE PROPAGATION LEMMA *Let $\mathcal{E}$ be safe for $\mathfrak{T}$ Let $R \in \mathcal{R}$ Let $\mathcal{T}$ be an $\mathcal{E}$-typing for $R \mid l$*

(i) *For any pattern node n of R and any $i \leq \mathrm{arity}(n)$*

$$\mathrm{TV}(\mathcal{T}(args(n)_i)) \subseteq \mathrm{TV}(\mathcal{T}(n))$$

(ii) *For any node $n \in R \mid l$*

$$\mathrm{TV}(\mathcal{T}(n)) \subseteq \mathrm{TV}(\mathcal{F}_{\mathcal{T}}(l))$$

PROOF (i) Straightforward

(ii) By (i) □

We can now show that any functional rewrite rule is typable according in the original sense (cf Definition 5 3 15)

5 3 32 PROPOSITION *Let $\mathcal{E}$ be safe for $\mathfrak{T}$ Let $R \in \mathcal{R}_{\mathcal{F}}$ Then R is typable*

PROOF Since $\mathcal{E}$ is principal there exists a balanced $\mathcal{E}$-typing for R, say $\mathcal{T}$, with $\mathcal{F}_{\mathcal{T}}(l) = \mathcal{E}(symb(l))$ We will show that $\mathcal{T} \restriction (R \mid l) = pt(R \mid l)$ Then we are done, since $\mathcal{T}$ is a rule typing Indeed, set $\mathcal{T}' = \mathcal{T} \restriction (R \mid l)$ Say $\mathcal{T}' = pt^*(R \mid l)$ Since $\mathcal{F}_{pt(R|l)}(l) \subseteq \mathcal{E}(symb(l))$ one has $\mathcal{F}_{\mathcal{T}'}(l) = \mathcal{F}_{pt(R|l)}(l)$ Now by the Variable Propagation Lemma (ii) the substitution $*$ is the identity function □

5 3 33 COROLLARY *Let $\mathfrak{T}$ be a TGRS Let $\mathcal{E}$ be an applicative type environment for $\mathfrak{T}$ If $\mathcal{E}$ is safe for $\mathfrak{T}$, then $\mathcal{E}$-typing satisfies the subject reduction property*

PROOF By Lemma 5 3 27 and Proposition 5 3 32 □

This immediately leads to a subject reduction result on $\mathcal{F}, \mathcal{A}$-typings, cf Theorem 5 3 24

5 3 34 DEFINITION Let $\mathcal{F}$ be a function type environment $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$

(i) $\mathcal{F}$ is called *Curry safe* for $\mathfrak{T}$ if for any Curry variant F_j occurring as pattern symbol in $\mathcal{R}_{\mathcal{F}}$ the following holds. Say $\mathcal{F}(F) = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$ Then

$$\mathrm{TV}(\sigma_1) \cup \cdots \cup \mathrm{TV}(\sigma_j) \subseteq \mathrm{TV}(\sigma_{j+1}) \cup \cdots \cup \mathrm{TV}(\sigma_{k+1}).$$

(ii) Let $\mathcal{A}$ be an algebraic type system. Then $\mathcal{F}$ is said to be *principal* for $\mathfrak{T}$ with respect to $\mathcal{A}$ if $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is principal for $\mathfrak{T}$.

5 3.35. THEOREM. *Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be applicative over $\mathcal{A}$. Let $\mathcal{F}$ be a function type environment for $\mathfrak{T}$. Suppose $\mathcal{F}$ is Curry safe and principal for $\mathfrak{T}$ (w.r.t $\mathcal{A}$). Then for any $g, h \in \mathcal{G}$*

$$\mathcal{F}, \mathcal{A} \vdash g : \sigma,\ g \to h \ \Rightarrow\ \mathcal{F}, \mathcal{A} \vdash h : \sigma.$$

PROOF. Let $\mathcal{F}$ is Curry safe and principal for $\mathfrak{T}$ (w.r.t. $\mathcal{A}$). First note that $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is applicative. Moreover $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is principal for $\mathfrak{T}$. Moreover $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is argument preserving by Curry safety of $\mathcal{F}$ and the restriction on variable occurrences in algebraic type definitions. Hence $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is safe for $\mathfrak{T}$ and we are done by Corollary 5.3.33. □

In the functional language *Miranda* (see Turner (1985)), only algebraic constructors are allowed as pattern symbols. Moreover, the type system of *Miranda* combines the systems of the type systems of Milner and Mycroft. Milner's type system is used for inference of types for functions for which no type is specified Mycroft's type system is applied to verify user-specified function types. This leads to a principal function type environment. Without describing the way to interpret *Miranda* scripts as graph rewrite systems, we conclude the following.

5.3.36. COROLLARY. *Typing in Miranda is preserved under evaluation*

Chapter 6

Uniqueness Typing in Graph Rewrite Systems

6.1. Introduction

The underlying motivation for uniqueness types was given by two fundamental problems in practical functional programming and the implementation of functional languages using sharing techniques.

The first problem is the *space behaviour* of functional programs during execution. In a reduction step one often has to construct complicated structures, involving creation of new nodes. One could improve the efficiency of the implementation by re-using the space of obsolete objects of the part of the graph being rewritten, thus performing garbage collection on the spot. It would even be better if one could predict *at compile time* which arguments of a function will become garbage during rewriting. This is called *compile time garbage collection*. This is often the only way to handle complex data structures efficiently.

A second issue is the incorporation of essentially non-functional operations in the formalism of graph rewriting, e.g. for dealing with input-output. File updating, for example, is an operation with side-effects, possibly disturbing referential transparency. Such operations are safe, however, if there exists only a single reference to the object being modified, at the moment the modification takes place.

The technique presented here involves incorporation of *locality* or *uniqueness* information in the type system mentioned above. Types are therefore extended with so called uniqueness attributes.

In the type of a function **F** it can now be indicated that a specific argument should be 'unique':

$$\mathbf{F} : (\sigma^{\bullet}, \ldots) \rightarrowtail \tau.$$

The intended meaning is that at the moment of evaluation, the corresponding object is local for **F**, i.e. can only be accessed via **F**. The type system will allow only applications of **F** of this kind.

This locality information can be used to solve the problems indicated above Suppose **F** has a rewrite rule If inspection of this rule yields that (part of) its unique argument is not used in the function result it can be concluded that this part becomes garbage in any concrete application It is then possible to reuse the space or even the contents of obsolete objects when building the result The second problem applies to external operations In this case, the solution is to declare the 'dangerous' updating functions in such a way that they require unique arguments

In order to achieve this, the notion of type assignment (for conventional typing) is extended the type correctness of an application $\mathbf{F}X$ depends not only on the argument type of **F** and the type of X, but also on the way X is passed to **F** if **F** expects a 'unique' argument then X should only be accessible by **F**, e g by requiring a reference count of 1 This straightforward reference count approach is rather rough In practice one usually has a specific evaluation order in mind In this paper we present a more liberal analysis using this information The correctness of a function application now depends on the demanded argument type, the offered argument type and the context in which the access takes place This dependency is formulated in terms of a subtyping relation specifying *coercions*

We prove that uniqueness typing is preserved under graph rewriting, for a sufficiently large class of graph rewrite systems

Related Work

The weighted reference count analysis, as presented in Section 6 3, is inspired by Guzmán and Hudak (1990) This paper addresses the mutability problem using a 'single threaded polymorphic lambda calculus' (*poly*-λ_{st}) It uses the operational semantics of lambda-graph reduction of Wadsworth (1971) In our paper the analysis is performed in the formalism of graph rewriting, which is obviously more direct The effect of cyclic structures (not present in the paper mentioned above) and general pattern matching are studied in the general graph rewriting setting presented here

In Wadler (1990), a type system including linear types is developed The paper also uses Wadsworth's lambda reduction

In Sastry et al (1993), the update problem is addressed by determining an order of evaluation of expressions via abstract interpretation such that destructive operators can be used instead of non-destructive ones In our approach the reduction order is restricted, and operators are either destructive or non-destructive The type system guarantees that all applications are safe Moreover, Sastry et al (1993) focus on a first-order call-by-value language with flat aggregates (i e aggregates only containing non-aggregate values) In the present paper, a higher-order call-by-need language is used with no specific assumptions on the structure of data

Based on the idea of uniqueness typing, Jacobs (1993) developed a *logical*

system explicitly mixing conventional and linear constructive logic. The approach described here is likely to offer a 'propositions-as-types' notion (representing proofs by typed graphs).

A variant of the type system described here has been incorporated in the lazy functional graph rewriting language *Clean*. So far, it has been used for the implementation of arrays and of an efficient high-level library for screen and file I/O (see Achten et al. (1993)).

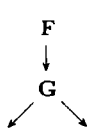
Structure of this Chapter

After an informal introduction to the technical aspects of uniqueness typing (Section 6.2), the reference analysis mentioned above is worked out in Section 6.3. The sections 6.4 and 6.5 extend the conventional type system with uniqueness information. Algebraic uniqueness types are constructed in Section 6.6, which also describes the uniqueness typing of higher-order functions. The sections 6.7 and 6.8 demonstrate that the reference analysis indeed captures the intended uniqueness property. The proof of the fact that typing is preserved during reduction is carried out stepwise in the sections 6.9–6.12. We give some examples in Section 6.13 and conclude with directions for future research in Section 6.14.

6.2. Informal Explanation

In this chapter, conventional typing will be extended with so called *uniqueness information*. This will enable us to indicate constraints on the *reference structure* of function-argument combinations. Consequently, the validity of an application of a function will depend on the *context* in which it appears, notably on the number of (external) references to its arguments.

This is best explained by an example. Suppose **F** is a unary operation, and we wish to require that in any application of **F**, the access of **F** to its argument is *unique*, i.e. the reference count of that argument object is 1. In our approach, this is done by dressing the conventional environment type of **F**, say $\sigma \rightarrowtail \tau$, with a uniqueness attribute on the argument type: $\sigma^{\bullet} \rightarrowtail \tau$. An application of **F** is then correct if the actual argument of **F** is of the right (conventional) type, and moreover the reference count of this object is 1. This is, however, not sufficient to maintain the correctness of the **F**-application: the uniqueness of **F**'s argument should be preserved during reduction.



In the above example, evaluation of **G** should not increase the reference count of **F**'s argument.

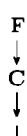
The idea is to encode this so called *access stability* in the result types of functions, again by a uniqueness attribute, e g $\mathbf{G} \quad (\rho_1, \rho_2) \rightarrowtail \sigma'^{\bullet}$ The typing requirements for rewrite rules will ensure that any application of such **G** is access stable the reference count of the result is not greater than the reference count of the application One could say that such an application is *potentially unique* uniqueness of the application and its reducts is guaranteed whenever the present reference count is 1

Observe that the potential uniqueness of an object can be handled in two ways, when offered as an argument of a function **F** The information can either be used (if **F** expects a unique argument, and the reference count if 1) or discarded (if **F** does not require any uniqueness, or the reference count is greater then 1) In the former case, the potentially unique object is *actually unique* for **F**, in the latter it *loses* its uniqueness This relation between offered and demanded type, determined by the reference situation, is expressed by *coercion relations* Roughly spoken, potentially unique objects can be coerced to actually unique or to non-unique ones

For some objects, however, one explicitly requires that they are *guaranteed unique* rather than just potentially unique This is the case for curried function applications We reserve the attribute $\triangle$ for such objects

Pattern matching and algebraic types

The treatment of functions using pattern matching (through algebraic constructors) is somewhat delicate Suppose **F** has a rewrite rule with the following pattern

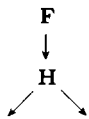


Moreover suppose we wish the argument of **C** in any matching part of a graph to be unique for **F**, i e there is only one path from **F** to this argument (say X), and X is only reachable via **F**

This requirement needs to be decomposed into referencewise dependencies, since an actual application of **F** might not yet be a redex, but could become one after some reduction steps It is needed to anticipate on future completion of the pattern

The desired property of X splits into two requirements X should be unique for **C**, and the **C**-node should be unique for **F** The idea is to encode this information entirely into the type of **F** both the uniqueness of $\perp$ and the uniqueness of **C** are indicated in the corresponding components of **F**'s argument type This looks like $\mathbf{F} \quad (\quad \cdot\sigma^{\bullet} \quad)^{\bullet} \rightarrowtail \tau$, where σ is the type of the $\perp$-argument The

result is that any partially matching application



has a unique first reference, and the type of the **H**-application is of the form $(\cdots\sigma'^{\bullet}\cdots)^{\bullet}$. The outermost uniqueness attribute expresses access stability; the innermost one ensures uniqueness of the second reference as soon as the pattern is completed.

The conventional types for algebraic constructors are furnished with uniqueness information. This leads to several uniqueness variants of these constructor types.

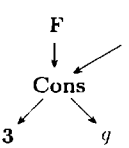
For the constructor **Cons** of lists, e.g., the possible variants include

Cons	:	$(\alpha, \mathrm{List}\ (\alpha)) \rightarrowtail \mathrm{List}\ (\alpha)$	(1)
Cons		$(\alpha, \mathrm{List}^{\bullet}(\alpha)) \rightarrowtail \mathrm{List}^{\bullet}(\alpha)$	(2)
Cons		$(\alpha^{\bullet}, \mathrm{List}\ (\alpha^{\bullet})) \rightarrowtail \mathrm{List}\ (\alpha^{\bullet})$	(3)
Cons	:	$(\alpha^{\bullet}, \mathrm{List}^{\bullet}(\alpha^{\bullet})) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{\bullet})$	(4)

With (1) ordinary lists can be built. (2) can be used for lists of which the 'spine' is unique, (3) for lists containing unique elements, and (4) for lists of which both the spine and the elements are unique.

The abovementioned encoding of uniqueness of 'deeper' arguments in the pattern of **F** is achieved by making the types of algebraic constructors *uniqueness propagating*: if one of their arguments is unique then the result is unique as well. This entails that the uniqueness of the $\perp$-argument is 'propagated upwards', leading to the uniqueness of the **C**-node. For the **Cons** example this means that variant (3) is rejected.

Uniqueness propagation imposes a restriction on the occurrences of type constructors: e.g. the type List(Int$^{\bullet}$) does not make sense. This is plausible if one realises that in an application



neither of the bottommost nodes can be considered as unique (when accessed through **Cons**) if the **Cons** node itself is not potentially unique. The argument of the type constructor List is said to be a *uniqueness propagating position*. The uniqueness propagating positions of type constructors are deduced by analyzing their algebraic definition.

The coercion relation mentioned above introduces a notion of *subtyping* on uniqueness variants Intuitively, a coercion statement $\sigma \leq \tau$ means that every σ-object can be regarded as a τ-object If the definition of T is

$$T\,\vec{\alpha} = C_1\,\vec{\sigma_1} \mid C_2\,\vec{\sigma_2} \mid \quad \cdot$$

then we would like

$$T\vec{\rho} \leq T\vec{\rho'} \qquad (*)$$

if and only if for any i

$$\vec{\sigma_i}[\vec{\alpha} := \vec{\rho}] \leq \vec{\sigma_i}[\vec{\alpha} := \vec{\rho'}]$$

The idea is that an object of type $T\vec{\rho}$ (say built by C_i) can be considered as a $T\vec{\rho'}$ object if the respective arguments of C_i can be coerced to each other The expression $(*)$ is reduced to componentwise coercion relations between $\vec{\rho}$ and $\vec{\rho'}$, determined by analyzing the occurrences of the variables $\vec{\alpha}$ in the algebraic specification of $T\vec{\alpha}$

Under special circumstances it is possible to use non-algebraic constructors in patterns of rewrite rules We will go into this in Section 6 6

Refined reference count and locality

The reference count approach described above is rather rough An object is considered non-unique if its reference count is greater than 1

In practice one often uses a graph rewrite system together with a specific reduction strategy The idea is that multiple references to a node are harmless if one knows that only one of them remains at the moment of evaluation E g the standard evaluation of a conditional statement **If** c **Then** t **Else** e causes first the evaluation of the c part, and subsequently evaluation of either t or e, but not both Hence, a single access to a node n in t combined with a single access to n in e would overall still result in a 'unique' access to n

This idea is generalized to arbitrary symbols by classifying the arguments according to the intended evaluation order Now suppose p, q are paths from m to n such that p and q are disjoint between start and end point Whether destructive access to n via p and q respectively is considered harmful depends on the argument classification of m

This contrasts the reference count approach which treats all references to a given node (the so called *access set* of that node) in the same way the uniqueness of an argument only depends on the size of the corresponding access set In the refined approach, some access references are considered harmful, others not This makes it necessary to distinguish between references in an access set In the paper we do this by a labelling mechanism

If the above p is indeed 'dangerous', destructive access to n via p will be prevented by *labelling* some reference in a tail part of p (containing only data nodes) as 'read-only' ($\otimes$) Possible 'write' access is indicated by $\odot$

The refined approach will be such that the access stability information (encoded in result types of functions) is still valid The notion of uniqueness is modified as follows A node n is unique for a node m if n is *local* for m, i e m is only reachable via n Note that this is indeed a weaker notion than the previous one It is, however, still usable to model destructive usage

For the reference analysis it is convenient if the root of the graph is not involved in the rewrite process For this reason we consider only TGRS's in which each object graph has the following standard shape the root has in degree 0 and contains a fixed data symbol **Root** of arity 1 (in Σ_0) For the analysis of other graphs g (notably the right-hand sides of rewrite rules) one temporarily attaches such an extra root to g The resulting graph is denoted as g^+

6.3. Usage Analysis

As mentioned before, one can refine the reference count approach by taking a specific reduction order into account As a first step, the idea is to characterize (as precisely as possible) an area in each graph containing the next redex to be contracted The evaluation process of a graph (leading to the selection of a redex) usually traverses the graph from its root downwards, directed by the function-argument structure This motivates the following classification, which translates the chosen evaluation directions into a (context independent) division of direct symbol arguments

6 3 1 DEFINITION (i) A *pre-classification* (for a symbol set Γ) is a function $\mathcal{P}$ such that for any S one has $\mathcal{P}(S) \subseteq \{1, \quad , \text{arity}(S)\}$

(ii) S is called *simple* if $\mathcal{P}(S) = \{1, \quad , \text{arity}(S)\}$

Arguments of S occurring in $\mathcal{P}(S)$ are called *primary arguments* of S The others are *secondary arguments* The above characterization can now be formalized by describing the 'active area' in a graph, consisting of nodes reachable from the root by descending via primary arguments These may appear as redex roots

6 3 2 DEFINITION Let g be a graph

(i) A reference $(n, i) \in Ref_g$ is a *primary reference* if i is a primary of $symb_g(n)$

(ii) A path p is a *primary path* if each reference on p is primary reference

(iii) A node $n \in g$ is a *primary node* if $p \quad r_g \rightsquigarrow n$ for some primary path p

(iv) Let $\Delta = \langle R, \mu \rangle$ be a redex in g Then Δ is a *primary redex* if $\mu(l)$ is a primary node The associated reduction relation is denoted by $\xrightarrow[p]{}$

In this paper we concentrate on $\xrightarrow[p]{}$ The idea is to classify multiple access within the active area as harmful Analogously multiple access within the non-active area is considered dangerous When a combination of accesses (active and non-active) occurs, only the active access is marked as unsafe

The analysis of the non-active accesses can be refined since in many cases (notably case distinctions c.q. conditionals) one can subdivide the secondary arguments into mutually exclusive groups such that accesses from different groups do not conflict.

6.3.3. DEFINITION. A *classification* for Γ is a pair $(\mathcal{P}, \mathcal{S})$ such that
(1) $\mathcal{P}$ is a pre-classification;
(2) each $\mathcal{S}(S)$ is a partition of the secondary arguments

$$\{1, \ldots, \text{arity}(S)\} \backslash \mathcal{P}(S).$$

We usually indicate the resulting sets by $\mathcal{P}^S, \mathcal{S}_1^S, \ldots, \mathcal{S}_n^S$, where n is the size of $\mathcal{S}(S)$.

A classification $(\mathcal{P}, \mathcal{S})$ induces a dependency relation on references and paths.

6.3.4. DEFINITION. (i) Let $S \in \Sigma$. Say S has arity ℓ. The relations $\sim_S$, $\lhd_S$ and $\lesssim_S$ on $\{1, \ldots, \ell\}$ are defined by

$$\begin{aligned} i \lhd_S j &\Leftrightarrow i \in \mathcal{P}^S \text{ and } j \notin \mathcal{P}^S, \\ i \sim_S j &\Leftrightarrow i, j \in \mathcal{P}^S \text{ or } i, j \in \mathcal{S}_k^S \text{ for some } k, \\ i \lesssim_S j &\Leftrightarrow i \lhd_S j \text{ or } i \sim_S j. \end{aligned}$$

(ii) Let g be a graph. The relation $\lesssim$ is defined on non-empty paths in g starting with the same node n, by

$$p \lesssim q \Leftrightarrow [p] \lesssim_{symb_g(n)} [q].$$

The dependency relation for direct arguments can be translated into a (global) dependency relation on all references in a graph.

6.3.5. DEFINITION. Let p, q be paths in g.
(i) p and q *diverge* (notation $p \wedge q$) if p, q start in the same node and are distinct elsewhere. More precisely, $p \wedge q$ if either $p = ()$ or $q = ()$ or

$$\mathrm{r}(p) = \mathrm{r}(q) \quad \text{and} \quad (\tilde{p})^- \# (\tilde{q})^-.$$

(ii) Let $a, b \in Ref_g$. By $(p, a) \wedge (q, b)$ we denote that $p \wedge q$ and p, q are extendible with a, b respectively.

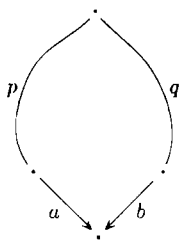
The relation $\lesssim$ on diverging paths induces the following relation on Ref_g and N_g respectively Intuitively, $a \lesssim b$ means that a might be used before b.

6.3.6. DEFINITION. Let g be a graph, and $a, b \in Ref_g$. Then

$$a \lesssim b \Leftrightarrow p * a \lesssim q * b \text{ for some acyclic } p, q \text{ with } (p, a) \wedge (q, b).$$

The dependency relation $\lesssim$ provides a weighted reference count analysis.

6.3.7. DEFINITION. A *critical path combination* is a quadruple p, a, q, b such that $(p, a) \wedge (q, b)$, the paths p, q are acyclic, $a \neq b$, and $\mathrm{d}(a) = \mathrm{d}(b)$



The common start node of $p * a$ and $q * b$ is called the *joining node* of the critical path combination.

Suppose $(p, a) \wedge (q, b)$ is a critical path combination. If $p * a \lesssim q * b$, then the reference a to $\mathrm{d}(a)$ might be used before b (in the $\lhd$ case) or a, b might be used in any order. The idea is now that $\mathrm{d}(a)$ is not allowed to be used destructively via $p * a$. This will be indicated by a suitable *labelling* of references with *use attributes* $\odot$ (for 'write use allowed') or $\otimes$ ('read access only').

A straightforward approach would label the reference a above with $\otimes$ (see Example 6.3.10). However, the usage information considered here is only important for *function* applications, in particular for parts of the graph matching the left-hand side of a rewrite rule. Since we consider systems with patterns (containing data nodes) we can be more liberal: in the above case it is sufficient that $p * a$ contains a reference labelled $\otimes$ anywhere in its 'data tail', to be made explicit below. The typing system will be such that this suffices to prevent destructive use of $\mathrm{d}(a)$ via the indicated path.

6.3.8. DEFINITION. (i) p is a *data path* if each $n \in p$ is a data node. (Note that $\mathrm{d}(p)$ needs not be a data node.)

(ii) The set of *use marking attributes* is $\mathbb{M} = \{\odot, \otimes\}$.

(iii) Let $use : Ref_g \to \mathbb{M}$ be a labelling. A path p in g is *marked* if there exist paths p_1, p_2 and a reference a such that $p = p_1 * a * p_2$, p_2 is a data path, and $use(a) = \otimes$.

(iv) A labelling *use* is a *marking* for g if for each critical path combination $(p, a) \wedge (q, b)$ one has

$$p * a \lesssim q * b \;\Rightarrow\; p * a \text{ is marked.}$$

Note that our analysis only considers non-overlapping paths, and does not distinguish data nodes from function nodes. We will not make any specific assumptions on the classification of data symbols, i.e. all data symbols are considered as simple. In order to assure that the reference analysis is stable during

reduction, we require that the classification is consistent with the way arguments are passed from the left-hand to the right-hand side of the rewrite rules in the TGRS in question.

6.3.9. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be an applicative TGRS. A *classification for* $\mathfrak{T}$ is a classification for $\Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T})$. This induces a classification for $\Sigma_{\mathfrak{T}}$ by regarding the data symbols and **Ap** as simple. It is required that the classification is consistent with each $R \in \mathcal{R}$, i.e. for each pair of border references a, b of R one has

$$a \lesssim b \text{ in } R \mid r \;\Rightarrow\; \bar{a} \lesssim b \text{ in } R \mid l.$$

Note that we have abstracted from any concrete reduction strategy and consider only $\xrightarrow[\mathcal{P}]{}$ reductions. In general one would like a suitable approximating classification for a given strategy in order to perform the analysis in this paper. Informally, a (one step) *reduction strategy* (for $\mathfrak{T}$) is a subset $\xrightarrow[s]{}$ of $\xrightarrow[\mathcal{R}]{}$. We say that a classification $(\mathcal{P}, \mathcal{S})$ *approximates* this strategy if $\xrightarrow[s]{} \subseteq \xrightarrow[\mathcal{P}]{}$. The trivial classification (considering all symbols simple) is of course always a safe approximation. However, it is desirable to have a classification such that $\mathcal{P}(S)$ is as small as possible, and subsequently $\mathcal{S}(S)$ consists of a maximal number of distinct classes. In practice this is difficult (or even impossible), but sometimes the way to define a strategy is close to an argument classification

The *functional reduction strategy* (used in *Clean*) is obtained by considering the following lazy evaluation procedure. First the rules are ordered During evaluation, the graph is examined from the root downwards If a function symbol is encountered, it will be tried to complete a pattern by reducing arguments of that function. The ordering of the rules gives preferences. If the pattern is completed, the appropriate rule is applied and evaluation continues. This gives a natural way for determining a classification: initially, the function arguments containing a non-$\bot$ pattern in $\mathcal{R}$ are classified as primary, and all others are in singleton secondary groups. The classification is completed by determining an '$\mathcal{R}$-consistent' closure via a fixed point construction. For the conditional **If** with the obvious rules

$$\begin{aligned} \mathbf{If}(\mathbf{True}.t, e) &\;\to\; t, \\ \mathbf{If}(\mathbf{False}, t, e) &\;\to\; e, \end{aligned}$$

this gives $\mathcal{P}^{\mathbf{If}} = \{1\}$, $\mathcal{S}^{\mathbf{If}}_1 = \{2\}$ and $\mathcal{S}^{\mathbf{If}}_2 = \{3\}$. The secondary classification consists of two classes since the rules are discarding either the second or the third argument.

There are two important examples of marking functions.

6.3.10. EXAMPLE. Let g be a graph

(i) 'Last reference' marking is done by defining use^{ℓ} as follows Let $n \in g$ Then for each $a \in acc_g(n)$

$$\begin{aligned} use^{\ell}(a) &= \otimes \quad \text{if } a \precsim b \text{ for some } b \in acc_g(n) \text{ with } b \neq a, \\ &= \odot \quad \text{otherwise} \end{aligned}$$

Note that this definition completely specifies the function use^{ℓ}

(ii) Straightforward reference counting is done by considering each symbol as simple and performing last reference marking More directly, this labelling is obtained by setting

$$\begin{aligned} use^{\mathrm{rc}}(a) &= \odot \quad \text{if d}(a) \text{ has reference count 1,} \\ &= \otimes \quad \text{otherwise} \end{aligned}$$

Using the standard classification of arguments of the conditional **If**, and no specific assumptions about other symbols, the above two examples give the following markings of the displayed graph

use^{ℓ} use^{rc}

6.4. Uniqueness Types

In this section we will extend the notion of typing (introduced in Section 5 3) with uniqueness information We therefore consider the set of types $\mathbb{T}$ and attach to each subtype a so called *uniqueness attribute* which may be $\times$ (for 'ordinary' or 'non-unique'), $\bullet$ (for 'unique') or $\triangle$ (for 'necessarily unique')

6 4 1 DEFINITION (i) The set of *uniqueness attributes* is $\mathbb{U} = \{\times, \bullet, \triangle\}$

(ii) The partial ordering $\leq$ on $\mathbb{U}$ is defined by

$$\bullet \leq \bullet, \quad \bullet \leq \times, \quad \times \leq \times, \quad \triangle \leq \triangle$$

This relation mirrors the intention that potentially unique objects may be used either as unique objects or as non-unique ones Necessarily unique objects remain unique

Uniqueness types are built from uniqueness variables, using the standard type constructor $\rightarrow$ and type constructors defined by an algebraic type system In the

sequel, let $\mathcal{A}$ be such an system. The uniqueness propagation mentioned in Section 6.2 restricts the applications of uniqueness variants of the type constructors in $\mathbb{C}_{\mathcal{A}}$. The dependency between the uniqueness of a type $T\vec{\sigma}$ and the uniqueness of the σ_i is determined by analyzing the dependencies in $\mathcal{A}$. Since $\mathcal{A}$ may contain (direct or indirect) recursion this is done by a fixedpoint construction.

We say that an occurrence of α in σ is *uniqueness propagating* if uniqueness of α induces uniqueness of σ. The idea is to define the notions 'uniqueness propagating occurrence' $uniocc_{\mathcal{A}}(\alpha, \sigma)$ and 'uniqueness propagating position' $uniprop_{\mathcal{A}}(T)_i$ simultaneously as the least solution of some predicate equations. This amounts to solving a least fixedpoint equation with respect to the ordering 'false' $\leq$ 'true'.

6.4.2. DEFINITION. The predicates $uniocc_{\mathcal{A}}$ and $uniprop_{\mathcal{A}}$ are specified by mutual induction, as follows. For the standard constructor $\rightarrow$ both $uniprop_{\mathcal{A}}(\rightarrow)_1$ and $uniprop_{\mathcal{A}}(\rightarrow)_2$ are set 'false'.

$$\begin{aligned} uniocc_{\mathcal{A}}(\alpha, \beta) &\Leftrightarrow \alpha = \beta, \\ uniocc_{\mathcal{A}}(\alpha, T(\sigma_1, \ldots, \sigma_k)) &\Leftrightarrow \bigvee_{i \leq k} uniprop_{\mathcal{A}}(T)_i \,\&\, uniocc_{\mathcal{A}}(\alpha, \sigma_i). \end{aligned}$$

Moreover for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$uniprop_{\mathcal{A}}(T)_i \Leftrightarrow \bigvee_{n} \bigvee_{j \leq \mathrm{arity}(C_n)} uniocc_{\mathcal{A}}(\alpha_i, \sigma_{nj}).$$

Using the predicates $uniprop_{\mathcal{A}}$, the collection of well-formed uniqueness types over $\mathcal{A}$ can be defined. E.g. List$^{\bullet}$(Int$^{\bullet}$) is correct, whereas List$^{\times}$(Int$^{\bullet}$) is not. The attribute $\vartriangle$ has the highest priority for propagation.

To simplify notation, we usually omit $\mathcal{A}$ in $uniprop_{\mathcal{A}}$ and $uniocc_{\mathcal{A}}$, and leave the algebraic system implicit.

6.4.3. DEFINITION. Let $u_1, \ldots, u_k \in \mathbb{U}$.

(i) Let $I \subseteq \{1 \ldots, k\}$. The *cumulative uniqueness attribute of* $\vec{u}$ *over* I (notation $\Sigma_I \vec{u}$) is defined by

$$\begin{aligned} \Sigma_I \vec{u} &= \vartriangle && \text{if } u_i = \vartriangle \text{ for some } i \in I; \text{ else:} \\ &= \bullet && \text{if } u_i = \bullet \text{ for some } i \in I, \\ &= \times && \text{otherwise.} \end{aligned}$$

We omit I if it is equal to $\{1 \ldots, k\}$.

(ii) Let $T \in \mathbb{C}_{\mathcal{A}}$ with arity k. The *T-relativized cumulative uniqueness attribute* of $\vec{u}$ (notation $\Sigma_T \vec{u}$) is $\Sigma_I \vec{u}$, where $I = \{i \mid uniprop(T)_i\}$.

6.4.4. DEFINITION. Let $T \in \mathbb{C}_{\mathcal{A}}$, say with arity k. Let $v, u_1, \ldots, u_k \in \mathbb{U}$. Then v is said to be *T-admissible* for $u_1, \ldots, u_k$ if $v = \Sigma_T \vec{u}$ whenever $\Sigma_T \vec{u} \neq \times$

6.4.5. DEFINITION. (i) For each attribute $u \in \mathbb{U}$ the set of *uniqueness variables over* u is defined by

$$\mathbb{V}^u = \{\alpha^u \mid \alpha \in \mathbb{V}\}.$$

The collection of uniqueness variables is

$$\hat{\mathbb{V}} = \mathbb{V}^\bullet \cup \mathbb{V}^\times \cup \mathbb{V}^\triangle.$$

(ii) For each $u \in \mathbb{U}$, the set of *uniqueness types over* u is defined by simultaneous induction as follows. Below, $u, v, u_1, \ldots$ range over $\mathbb{U}$, and T over $\mathbb{C}_A$.

$$\begin{array}{rcl} \alpha \in \mathbb{V}^u & \Rightarrow & \alpha \in \mathbb{T}^u, \\ \sigma \in \mathbb{T}^v, \tau \in \mathbb{T}^w & \Rightarrow & \sigma \xrightarrow{u} \tau \in \mathbb{T}^u, \\ \left.\begin{array}{r} \sigma_1 \in \mathbb{T}^{u_1}, \ldots, \sigma_k \in \mathbb{T}^{u_k}, \\ u \text{ is } T\text{-admissible for } u_1, \ldots, u_k \end{array}\right\} & \Rightarrow & T^u(\sigma_1, \ldots, \sigma_k) \in \mathbb{T}^u. \end{array}$$

Set $\hat{\mathbb{T}} = \mathbb{T}^\times \cup \mathbb{T}^\bullet \cup \mathbb{T}^\triangle$.

(iii) For each $\sigma \in \hat{\mathbb{T}}$ the *attribute* of σ (notation $[\sigma]$) is defined by

$$[\sigma] = u \text{ if } \sigma \in \mathbb{T}^u.$$

(iv) The set of *uniqueness symbol types* (notation $\hat{\mathbb{T}}_S$) is defined by

$$\sigma_1, \ldots, \sigma_k, \tau \in \hat{\mathbb{T}} \Rightarrow (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau \in \hat{\mathbb{T}}_S.$$

(v) For $\sigma \in \hat{\mathbb{T}}, \hat{\mathbb{T}}_S$, let $|\sigma|$ denote the *stripped version* of σ, i.e. σ without any uniqueness attributes. Note that $|\sigma| \in \mathbb{T}, \mathbb{T}_S$.

NOTATION. Let $\vec{\sigma} \in \hat{\mathbb{T}}$. Then $\Sigma\vec{\sigma}$ denotes the cumulative uniqueness attribute $\Sigma[\vec{\sigma}]$. Analogously we use the T-relativized version.

As was mentioned in Section 6.2, the description of type assignment uses coercion relations, depending on the kind of reference via which an argument is accessed by a function or constructor.

We introduce coercion relations for access via $\odot$-references and $\otimes$-references respectively. The idea is that unique objects keep their uniqueness while being passed via $\odot$-arcs, and lose it when they are accessed via $\otimes$-references. Furthermore, a unique object can be coerced to a non-unique one (if such a non-unique object is demanded in a function application). Necessarily unique objects remain unique in all cases.

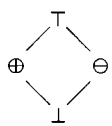
Intuitively, a coercion statement $\sigma \leq \tau$ means that every σ-object can be regarded as a τ-object. For function types this view leads to the rule

$$\sigma' \leq \sigma, \tau \leq \tau' \Rightarrow \sigma \to \tau \leq \sigma' \to \tau'.$$

Note the so called contravariance in the first argument of the type constructor $\rightarrow$: one says that the first argument occurs on a *negative* position. This is generalized to arbitrary constructors: the coercion rules are made dependent on a 'negative/positive' classification of constructor arguments.

First we will explain how an algebraic type system $\mathcal{A}$ determines a classification function $sign_{\mathcal{A}}$ for type constructors. The idea is that the classification of a type constructor is deduced from the syntactical form of the types in each clause of its algebraic definition. The technique is analogous to the specification method for *uniprop*.

6.4.6. DEFINITION. (i) The *sign classification lattice* $\mathbb{S}$ is the set $\{\bot, \oplus, \ominus, \top\}$, partially ordered by $\sqsubseteq$, as follows.



(ii) The binary operation $\cdot$ on $\mathbb{S}$ is defined as follows. Here s ranges over $\mathbb{S}$.

$$\begin{aligned} \bot \cdot s = s \cdot \bot &= \bot, \\ \top \cdot s = s \cdot \top &= \top \quad \text{if } s \neq \bot, \\ \oplus \cdot \oplus = \ominus \cdot \ominus &= \oplus, \\ \oplus \cdot \ominus = \ominus \cdot \oplus &= \ominus. \end{aligned}$$

The classification of type constructor arguments is determined using an auxiliary function *occ*, determining the kind of occurrences of each variable in a given type.

6.4.7. DEFINITION. The functions occ $(= occ_{\mathcal{A}}) : \mathbb{V} \times \mathbb{T} \rightarrow \mathbb{S}$ and $sign$ $(= sign_{\mathcal{A}})$ are specified below by simultaneous induction. For the moment we consider $\rightarrow$ as an ordinary constructor with $sign(\rightarrow)_1 = \ominus$ and $sign(\rightarrow)_2 = \oplus$.

$$\begin{aligned} occ(\alpha, \alpha) &= \oplus, \\ occ(\alpha, \beta) &= \bot \quad \text{if } \beta \neq \alpha, \\ occ(\alpha, T(\sigma_1, \ldots, \sigma_k)) &= \bigsqcup_{i \leq k} sign(T)_i \cdot occ(\alpha, \sigma_i) \end{aligned}$$

Moreover for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$sign(T)_i = \bigsqcup_n \bigsqcup_{j \leq \mathrm{arity}(C_n)} occ(\alpha_i, \sigma_{nj}).$$

6.4.8. EXAMPLE. If $\mathcal{A}$ contains

$$\begin{aligned} \mathrm{List}(\alpha) &= \mathbf{Cons}(\alpha, \mathrm{List}(\alpha)) \mid \mathbf{Nil}, \\ \mathrm{T}(\alpha, \beta, \gamma) &= \mathbf{C}(\mathrm{List}(\beta \to \alpha)) \mid \mathbf{D}(\gamma \to \mathrm{List}(\gamma)), \end{aligned}$$

then

$$\begin{aligned} \mathit{sign}(\mathrm{List}) &= \oplus, \\ \mathit{sign}(\mathrm{T}) &= (\oplus, \ominus, \top). \end{aligned}$$

We say that α *occurs essentially* in σ if $\mathit{occ}(\alpha, \sigma) \neq \bot$. If T is a constructor with $\mathit{sign}(T)_i = \bot$, then apparently α_i does not occur essentially in any of the clauses in the definition of $T\vec{\alpha}$. Observe that this happens if α_i either is absent in these clauses or occurs only in a (direct or indirect) recursion of T. Thus the i-th position of a type $T\vec{\sigma}$ is non-essential. Without loss of generality, we can assume that such non-essential positions of type constructors T do not exist: this does not impose a serious restriction on the expressive power of the system. An important consequence is the following.

6.4.9. LEMMA. $\mathit{occ}(\alpha, \sigma) = \bot \;\Rightarrow\; \alpha \notin \mathrm{TV}(\sigma)$.

The classifications *sign* and *uniprop* are related in the following way.

6.4.10. LEMMA. $\mathit{uniprop}(T)_i \;\Rightarrow\; \mathit{sign}(T)_i \sqsupseteq \oplus$.

Now the coercion properties of constructor applications are defined using the deduced classification of constructor arguments. For convenience we introduce the following notation.

NOTATION. Let A be a set, and R a relation on A.

(i) For each $s \in \mathbb{S}$, the s-variant of R is defined by

$$\begin{aligned} x\ {}^{\oplus}R\ y \quad &\text{if} \quad x\ R\ y, \\ x\ {}^{\ominus}R\ y \quad &\text{if} \quad y\ R\ x, \\ x\ {}^{\top}R\ y \quad &\text{if} \quad x\ {}^{\oplus}R\ y \text{ and } x\ {}^{\ominus}R\ y, \\ (x\ {}^{\bot}R\ y \quad &\text{if} \quad x\ {}^{\oplus}R\ y \text{ or } x\ {}^{\ominus}R\ y). \end{aligned}$$

(ii) This denotation naturally extends to sequences of types:

$$\vec{x}\ {}^{\vec{s}}R\ \vec{y} \quad \text{if} \quad x_i\ {}^{s_i}R\ y_i \text{ for each } i.$$

6.4.11. DEFINITION. The *general coercion relation* $\leq$ on $\hat{\mathbb{T}}$ is defined inductively as follows.

$$\begin{gathered} \alpha^u \leq \alpha^u, \\ u \leq u',\ \ \vec{\sigma}\ {}^{\mathit{sign}(T)}\!\leq \vec{\sigma'} \;\Rightarrow\; T^u\vec{\sigma} \leq T^{u'}\vec{\sigma'}. \end{gathered}$$

Coercions along $\odot$ and $\otimes$ references can now be made explicit.

6.4.12. DEFINITION. The coercion relations $\leq^{\odot}$ and $\leq^{\otimes}$ are defined by setting

$$\begin{aligned} \sigma \leq^{\odot} \tau &\Leftrightarrow \sigma \leq \tau, \\ \sigma \leq^{\otimes} \tau &\Leftrightarrow \sigma \leq \tau \text{ and } [\tau] = \times. \end{aligned}$$

In this paper we will consider uniqueness symbol types which are *uniformly attributed* variants of conventional symbol types, i.e. throughout a type, uniqueness attributes of variables are equal whenever the underlying variables are. This is no essential restriction but will simplify the description and analysis of the system.

6.4.13. DEFINITION. (i) Let $\sigma \in \mathbb{T}$. A *variable attribute environment* for σ is a function $\varphi : \mathrm{TV}(\sigma) \to \mathbb{U}$.

(ii) Let $\sigma \in \hat{\mathbb{T}}, \hat{\mathbb{T}}_S$. Then φ_σ denotes the corresponding variable attribute environment for $|\sigma|$. This is uniquely determined if σ is uniformly attributed.

Uniform attribution enables us to view substitutions as follows.

6.4.14. DEFINITION. (i) A *substitution* is a function $* : \mathbb{V} \to \hat{\mathbb{T}}$.

(ii) $*$ is a *substitution for* σ if it respects the attributes in σ, i.e. for each $\alpha \in \mathrm{TV}(|\sigma|)$

$$[*(\alpha)] = \varphi_\sigma(\alpha).$$

The result of applying $*$ to σ is denoted by σ^*.

(iii) The notion of *instance* ($\subseteq$) is modified accordingly.

The coercions are defined in such a way that the expected substitutivity results hold.

6.4.15. LEMMA. (i) $\leq^{\odot}$ *is reflexive on* $\hat{\mathbb{T}}$.

(ii) $\leq^{\otimes}$ *is reflexive on* $\mathbb{T}^{\times}$.

PROOF. Easy induction □

6.4.16. LEMMA. *Let* $\sigma, \tau \in \hat{\mathbb{T}}$, *and let* $*$ *be a substitution.*

(i) $\sigma \leq^{\odot} \tau \Rightarrow \sigma^* \leq^{\odot} \tau^*$.

(ii) $\sigma \leq^{\otimes} \tau \Rightarrow \sigma^* \leq^{\otimes} \tau^*$.

PROOF. By induction on the generation of the coercion relations, using Lemma 6.4.15 in the variable-to-variable case. □

6.5. Type Assignment

In our new environments we allow symbols to have more than one type. This is needed, e.g., for incorporation of multiple variants of the types of algebraic constructors, and for the basic symbols.

6.5.1. DEFINITION. (i) Let $\Gamma \subseteq \Sigma$ be a set of symbols. A *uniqueness type environment for* Γ is a function $\mathcal{E} : \Gamma \to \wp(\widehat{\mathbb{T}}_S)$ such that for any $S \in \Gamma$ one has the following.

(1) Each $\sigma \in \mathcal{E}(S)$ is uniformly attributed.

(2) $|\sigma| = |\sigma'|$ for all $\sigma, \sigma' \in \mathcal{E}(S)$.

(ii) A *uniqueness types environment for* $\mathfrak{T}$ is an environment for $\Sigma_{\mathfrak{T}}$.

(iii) The *basic uniqueness type environment* $\mathcal{E}_0$ is the following environment for Σ_0.

$$\begin{aligned} \mathcal{E}_0(\bot) &= \{\alpha^u \mid u \in \mathbb{U}\}, \\ \mathcal{E}_0(\mathbf{Ap}) &= \{(\alpha^s \xrightarrow{t} \beta^u, \alpha^s) \rightarrowtail \beta^u \mid s, u \in \{\times, \bullet, \vartriangle\}, t \in \{\vartriangle, \times\}\}, \\ \mathcal{E}_0(\mathbf{Root}) &= \{\alpha^u \rightarrowtail \alpha^u \mid u \in \mathbb{U}\}. \end{aligned}$$

(iv) For any uniqueness environment $\mathcal{E}$, the underlying conventional environment is denoted by $|\mathcal{E}|$

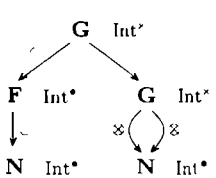
6.5.2. DEFINITION. Let $g = \langle N, symb, args \rangle$ be a graph, and let $\mathcal{E}$ be a uniqueness type environment.

(i) Let $\mathcal{U} : N \to \widehat{\mathbb{T}}$ be a uniqueness type assignment, and *use* a marking function for g. Then $\mathcal{U}$ is an *$\mathcal{E}$-uniqueness typing for g according to use* if for each $n \in g$ there exist $\sigma \in \mathcal{E}(symb(n))$ and $\tau_1, \ldots, \tau_k \in \widehat{\mathbb{T}}$ such that

$$\begin{gathered} \mathcal{U}(args(n)_i) \leq^{use(n)_i} \tau_i \quad \text{for any } i \leq k = \text{arity}(n), \\ (\tau_1, \ldots, \tau_k) \rightarrowtail \mathcal{U}(n) \subseteq \sigma. \end{gathered}$$

(ii) g is *$\mathcal{E}$-typable with type* σ (notation $\mathcal{E} \vdash g \cdot \sigma$) if there exists a marking function *use* and a uniqueness typing $\mathcal{U}$ for g according to *use* such that $\mathcal{U}(r_g) = \sigma$. We write $\mathcal{E} \vdash_{use} g \cdot \sigma$ if we wish to indicate the applied marking explicitly.

6.5.3. EXAMPLE. The following gives (parts of) a well-typed graph and the corresponding environment.



F	:	$\alpha^\bullet \rightarrowtail \alpha^\bullet$,
G	:	$(\beta^\times, \beta^\times) \rightarrowtail \beta^\times$,
N	:	$\mathbf{Int}^\bullet$

6.5.4. DEFINITION. Let $g = \langle N, symb, args\rangle$ be a graph, and $\mathcal{E}$ be an environment. Furthermore, let $\mathcal{U} : N \to \hat{\mathbb{T}}$ be a uniqueness type assignment.

(i) Let $n \in g$. The *plain function type* of n according to $\mathcal{U}$ (notation $\mathcal{F}_{\mathcal{U}}(n)$) is

$$\mathcal{F}_{\mathcal{U}}(n) = (\mathcal{U}(n_1), \ldots, \mathcal{U}(n_k)) \rightarrowtail \mathcal{U}(n),$$

where $k = \text{arity}(n)$, and $n_i = args(n)_i$.

(ii) $\mathcal{U}$ is an *plain $\mathcal{E}$-uniqueness typing for* g if for each $n \in g$ there exists $\sigma \in \mathcal{E}(symb(n))$ such that

$$\mathcal{F}_{\mathcal{U}}(n) \subseteq \sigma.$$

Again, we will omit $\mathcal{E}$ if it is clear from the context.

The notion of uniqueness typing for rewrite rules distinguishes functional and Curry rules. The presentation is inspired by the analysis at the end of Section 5.3.

6.5.5. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R}\rangle$ be a TGRS. Let R be a rewrite rule.

(i) A *marking function for* R is a marking function for $(R \mid r)^+$. If *use* is a marking function for R then the *root mark* of R (notation $use(R)$) is the value of *use* in the root reference of $(R \mid r)^+$.

(ii) Let *use* be a marking for R. Let $\sigma \in \hat{\mathbb{T}}_S$. A function $\mathcal{U} : g_R \to \hat{\mathbb{T}}$ is an *$\mathcal{E}$-uniqueness typing* for R (according to *use*) if the following requirements are satisfied

(1) $\mathcal{U}$ is a plain uniqueness typing for $R \mid l$,

(2) $\mathcal{U}$ is a uniqueness typing (according to *use*) for $R \mid r$,

(3) $\mathcal{U}(r) \leq^{use(R)} \mathcal{U}(l)$.

(iii) R is *$\mathcal{E}$-typable with function type* $\vec{\sigma} \rightarrowtail \tau$ (notation $\mathcal{E} \vdash R : \vec{\sigma} \rightarrowtail \tau$) if there exist a marking *use* and a typing $\mathcal{U}$ for R according to *use* such that $\mathcal{F}_{\mathcal{U}}(l) = \vec{\sigma} \rightarrowtail \tau$.

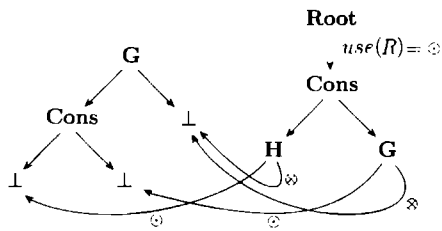
(iv) A rule R for F is *$\mathcal{E}$-uniqueness typable* if $\mathcal{E} \vdash R : \sigma$ for each $\sigma \in \mathcal{E}(F)$.

(v) $\mathcal{R}$ is *uniqueness typable* if each $R \in \mathcal{R}_{\mathcal{F}}$ is $\mathcal{E}$-uniqueness typable.

Some explanations are in place. The left-hand side is to be typed plainly, to establish the encoding of uniqueness of 'deeper' arguments into the function type (cf. Section 6.2).

The coercion condition (3) for r is included to account for the effect of redirection in the construction of the contractum. This will become clear in the proof of the subject reduction property for this system. Moreover the marking function for R is only essential in the case of an extending rewrite rule.

6.5.6. EXAMPLE. Consider the following (recursive) rule for **G**.



This rule is uniqueness typable using the following environment.

$$\begin{array}{rcl} \mathbf{G} & : & (\mathrm{List}^{\bullet}(\alpha^{\times}), \alpha^{\times}) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{\times}), \\ \mathbf{H} & : & (\alpha^{\times}, \alpha^{\times}) \rightarrowtail \alpha^{\times}, \\ \mathbf{Cons} & : & (\alpha^{\times}, \mathrm{List}^{\bullet}(\alpha^{\times})) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{\times}). \end{array}$$

6.6. Uniqueness Type Environments

In this section we will give two standard constructions for type environments concerning data symbols: an environment $\mathcal{E}_{\mathcal{A}}$ for algebraic constructors introduced in $\mathcal{A}$, and an environment $\mathcal{E}_{\mathcal{F}}$ for function symbols and their Curry variants.

In order to describe coercion and uniqueness properties of environments we introduce some terminology.

6.6.1. DEFINITION. (i) The *schematic coercion relation* $\precsim$ is defined as $\leq$ (see Definition 6.4.11), extended with the clause

$$u \leq v \;\Rightarrow\; \alpha^u \precsim \alpha^v.$$

(ii) Let $\sigma_1, \sigma_2 \in \hat{\mathbb{T}}$, each uniformly attributed. Suppose $|\sigma_1| = |\sigma_2|$ (= σ, say). Let $*_1, *_2$ be substitutions for σ_1, σ_2 respectively. The *combined coercion relation* is defined by

$$(\sigma_1, *_1) \leq (\sigma_2, *_2) \;\Leftrightarrow\; \sigma_1 \precsim \sigma_2 \text{ and } \forall \alpha \in \sigma \;\; [*_1(\alpha) \overset{occ(\alpha,\sigma)}{\leq} *_2(\alpha)].$$

6.6.2. LEMMA. *Let $\sigma_1, \sigma_2 \in \hat{\mathbb{T}}$, each uniformly attributed. Suppose $|\sigma_1| = |\sigma_2|$. Furthermore, let $*_1, *_2$ be substitutions for σ_1, σ_2. Then*

$$\sigma_1^{*_1} \leq \sigma_2^{*_2} \;\Leftrightarrow\; (\sigma_1, *_1) \leq (\sigma_2, *_2).$$

PROOF. By induction on the structure of $\sigma = |\sigma_1|$.

The case $\sigma = \alpha$ is easy.

Suppose $\sigma = T(\tau_1, \ldots, \tau_k)$. Say $\sigma_1 = T^{u_1}(\tau_{11}, \ldots, \tau_{1k})$, $\sigma_2 = T^{u_2}(\tau_{21}, \ldots, \tau_{2k})$, and $s_i = sign(T)_i$.

($\Rightarrow$) By assumption

$$u_1 \leq u_2 \text{ and } (\tau_{1i})^{*_1} \overset{s_i}{\leq} (\tau_{2i})^{*_2}.$$

Using the induction hypothesis one obtains

$$(\tau_{1i}, *_1) \overset{s_i}{\leq} (\tau_{2i}, *_2).$$

Hence $(T^{u_1}(\tau_{11}, \ldots, \tau_{1k}), *_1) \leq (T^{u_2}(\tau_{21}, \ldots, \tau_{2k}), *_2)$
($\Leftarrow$) Similarly. □

6.6.3. DEFINITION. Let $\mathcal{E}$ be a type environment, and let $S \in \Sigma$.

(i) $\mathcal{E}$ is *coercion consistent* w.r.t. S if for all $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(S)$ one has

$$\tau \precsim \tau' \;\Rightarrow\; \vec{\sigma} \precsim \vec{\sigma'}.$$

(ii) A symbol type $(\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ is *occurrence increasing* if for all $\alpha \in \mathbb{V}$ and $i \leq k$

$$occ(\alpha, \sigma_i) \sqsubseteq occ(\alpha, \tau).$$

(iii) $\mathcal{E}$ is *occurrence increasing* w.r.t. S if $|\mathcal{E}|(S)$ is occurrence increasing.

(iv) $\mathcal{E}$ is *coercion safe* w.r.t. S if $\mathcal{E}$ is both coercion consistent and occurrence increasing w.r.t. S.

(v) $\mathcal{E}$ is *uniqueness propagating* w.r.t. S if for any $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}(S)$

$$\Sigma\vec{\sigma} \neq \times \;\Rightarrow\; [\tau] \neq \times.$$

(vi) $\mathcal{E}$ is *uniqueness safe* w.r.t. S if $\mathcal{E}$ is coercion safe and uniqueness propagating w.r.t. S.

(vii) Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS. $\mathcal{E}$ is *uniqueness safe* for $\mathfrak{T}$ if $\mathcal{E}$ is uniqueness safe with respect to all pattern symbols appearing in $\mathcal{R}_{\mathcal{F}}$.

6.6.4. REMARK. Let $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ be uniformly attributed and occurrence increasing. Then any substitution for τ is a substitution for the σ_i by Lemma 6.4.9.

Coercion safety for environment types implies the following coercion property for their instances.

6.6.5. PROPOSITION. *Let $\mathcal{E}$ be coercion safe w.r.t. S. Let $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(S)$. Then for any pair $*, *'$ of substitutions for τ, τ' and any i*

$$\tau^* \leq (\tau')^{*'} \;\Rightarrow\; (\sigma_i)^* \leq (\sigma'_i)^{*'}.$$

PROOF. Let $*, *'$ be substitutions. Then for any i

$$\begin{aligned} \tau^* \leq (\tau')^{*'} \quad &\Rightarrow \quad (\tau, *) \leq (\tau', *'), \quad \text{by Lemma 6.6.2} \\ &\Rightarrow \quad (\sigma_i, *) \leq (\sigma'_i, *'), \quad \text{by coercion safety} \\ &\Rightarrow \quad \sigma_i^* \leq (\sigma'_i)^{*'}, \quad \text{by Lemma 6.6.2.} \ \square \end{aligned}$$

Algebraic type environments

In our treatment of algebraic type systems, we will assume that the definitions are *monotonic* when viewed as inductive operators. This corresponds to the semantical intuition where algebraic definitions are viewed as inductive definitions. This boils down to the assumption that in any definition of the form

$$T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$$

all occurrences the defined constructor T in the σ_i are positive. It is possible, however, to admit non-monotonic algebraic definitions if one restricts the coercion relation for constructor types. This will not be worked out here.

We will now explain how the types of algebraic constructors are derived from an algebraic type system. One could consider the types that are obtained by consistently attributing the variables and constructor symbols in an algebraic definition. For lists this would result in nine uniqueness variants for **Cons**:

Cons	:	$(\alpha^{\times}, \mathrm{List}^{\times}(\alpha^{\times})) \rightarrowtail \mathrm{List}^{\times}(\alpha^{\times})$	(1)
Cons		$(\alpha^{\times}, \mathrm{List}^{\bullet}(\alpha^{\times})) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{\times})$	(2)
Cons		$(\alpha^{\bullet}, \mathrm{List}^{\times}(\alpha^{\bullet})) \rightarrowtail \mathrm{List}^{\times}(\alpha^{\bullet})$	(3)
Cons		$(\alpha^{\bullet}, \mathrm{List}^{\bullet}(\alpha^{\bullet})) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{\bullet})$	(4)
Cons	:	$(\alpha^{\triangle}, \mathrm{List}^{\triangle}(\alpha^{\triangle})) \rightarrowtail \mathrm{List}^{\triangle}(\alpha^{\triangle})$	(5)
	⋮		

Since the type constructor list List is uniqueness propagating in its argument, some of the combinations (e.g. (3)) are illegal.

We will make the above idea more explicit by giving a method for consistently attributing the constructor types associated with an algebraic type definition of $T\vec{\alpha}$. The starting point will be an attribute assignment to the variables $\vec{\alpha}$, and an admissible attribute t for T. The idea is to attribute all type constructors with $\times$, unless another attribute is necessary by propagation considerations. An exception is made for direct recursion, i.e. for the occurrences of the defined constructor T. This allows e.g. the construction of spine-unique lists.

6.6.6. DEFINITION. Let T be a type constructor with arity k.

(i) A $T\vec{\alpha}$ attribution is a pair consisting of a variable attribute environment φ for $\vec{\alpha}$ and an attribute t such that t is T-admissible for $\varphi(\vec{\alpha})$.

(ii) For each $T\vec{\alpha}$ attribution φ, t, and each $\sigma \in \mathbb{T}(\vec{\alpha})$, the the *$T$-type attribution* of σ (notation $[\![\sigma]\!]^T_{\varphi,t}$) is defined as follows.

$$\begin{aligned} [\![\alpha]\!]^T_{\varphi,t} &= \alpha^{\varphi(\alpha)}, \\ [\![T(\sigma_1, \ldots, \sigma_k)]\!]^T_{\varphi,t} &= T^u([\![\sigma_1]\!]^T_{\varphi,t}, \ldots, [\![\sigma_k]\!]^T_{\varphi,t}), \end{aligned}$$

$$\begin{array}{rcll} \text{where } u & = & \triangle & \text{if } \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} = \triangle \text{ or } t = \triangle, \text{ else:} \\ & = & \bullet & \text{if } \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} = \bullet \text{ or } t = \bullet, \\ & = & \times & \text{otherwise.} \end{array}$$

$$[\![U(\sigma_1,\ldots,\sigma_k)]\!]^T_{\varphi,t} = U^u([\![\sigma_1]\!]^T_{\varphi,t},\ldots,[\![\sigma_k]\!]^T_{\varphi,t}), \quad \text{where } u = \Sigma_U[\![\vec{\sigma}]\!]^T_{\varphi,t}.$$

Note that $[\![\sigma]\!]^T_{\varphi,t}$ is well defined for all appropriate σ, i.e. all attributes assigned to type constructors are admissible.

As with conventional types, a set $\mathcal{A}$ of algebraic type definitions induces a uniqueness type environment $\mathcal{E}_{\mathcal{A}}$ for all constructors. This is done in the following way.

6.6.7. DEFINITION. The *uniqueness type environment $\mathcal{E}_{\mathcal{A}}$ associated with $\mathcal{A}$* is defined by setting for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$\mathcal{E}_{\mathcal{A}}(C_n) = \{[\![\vec{\sigma_n} \rightarrowtail T\vec{\alpha}]\!]^T_{\varphi,t} \mid (\varphi,t) \text{ is a } T\vec{\alpha} \text{ attribution}\}.$$

Observe that $\mathcal{E}_{\mathcal{A}}$ is indeed a proper uniqueness type environment (see Definition 6.5.1).

In the remainder of this subsection we will show that $\mathcal{E}_{\mathcal{A}}$ is both coercion safe and uniqueness propagating.

6.6.8. PROPOSITION. *$\mathcal{E}_{\mathcal{A}}$ is occurrence increasing.*

PROOF. This follows directly from the way the functions *sign* and *occ* are determined by the algebraic type system $\mathcal{A}$. □

We now analyze the variants for the clauses in the definition of $T\vec{\alpha}$ in $\mathcal{A}$, obtained by the attribution function $[\![\ ]\!]^T$.

6.6.9. DEFINITION. For each $\sigma \in \mathbb{T}(\vec{\alpha})$, the relation $\leq^{\sigma,T}$ on $T\vec{\alpha}$ attributions is defined as follows.

$$(\varphi,t) \leq^{\sigma,T} (\varphi',t') \Leftrightarrow \begin{cases} \forall\alpha \in \sigma\ [\varphi(\alpha) \overset{occ(\alpha,\sigma)}{\leq} \varphi'(\alpha)], \\ t \overset{occ(T,\sigma)}{\leq} t'. \end{cases}$$

6.6.10. LEMMA. $(\varphi,t) \leq^{\sigma,T} (\varphi',t') \Rightarrow [\![\sigma]\!]^T_{\varphi,t} \precsim [\![\sigma]\!]^T_{\varphi',t'}$.

PROOF. By induction on the structure of σ.

If $\sigma = \alpha$ this is simple.

Suppose $\sigma = T(\sigma_1,\ldots,\sigma_k)$, and $(\varphi,t) \leq^{\sigma,T} (\varphi',t')$ By definition of *occ* one has $(\varphi,t) \overset{sign(T)_i}{\leq}{}^{\sigma_i,T} (\varphi',t')$ for each $i \leq k$. Hence by the induction hypothesis (k times)

$$[\![\sigma_i]\!]^T_{\varphi,t} \overset{sign(T)_i}{\precsim} [\![\sigma_i]\!]^T_{\varphi',t'}.$$

It remains to show that $[[\![T\vec{\sigma}]\!]^T_{\varphi,t}] \leq [[\![T\vec{\sigma}]\!]^T_{\varphi',t'}]$. By the induction hypothesis and Lemma 6.4.10 one has $\Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} \leq \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi',t'}$. Combined with $t \leq t'$ this yields the result.

The case $\sigma = U(\sigma_1,\ldots,\sigma_k)$ with $U \neq T$ is treated similarly. □

6.6.11. PROPOSITION. *Let $C\vec{\sigma}$ be a clause in the definition of $T\vec{\alpha}$. Then for any $T\vec{\alpha}$ attributions (φ, t) and (φ', t') one has*

$$[\![T\vec{\alpha}]\!]^T_{\varphi,t} \precsim [\![T\vec{\alpha}]\!]^T_{\varphi',t'} \Rightarrow [\![\sigma_i]\!]^T_{\varphi,t} \precsim [\![\sigma_i]\!]^T_{\varphi',t'}.$$

PROOF. Suppose $[\![T\vec{\alpha}]\!]^T_{\varphi,t} \precsim [\![T\vec{\alpha}]\!]^T_{\varphi',t'}$. Then $(\varphi, t) \leq^{T\vec{\alpha},T} (\varphi', t')$. Let $i \leq k$. By construction of *occ*, *sign* and the fact that T occurs only positively in σ_i one has $(\varphi, t) \leq^{\sigma_i,T} (\varphi', t')$, and we are done by Lemma 6.6.10. □

6.6 12. COROLLARY. (i) *$\mathcal{E}_{\mathcal{A}}$ is coercion consistent.*
(ii) *$\mathcal{E}_{\mathcal{A}}$ is coercion safe*

It remains to show that algebraic type environments are uniqueness propagating.

6.6.13. LEMMA. *Let (φ, t) be a $T\vec{\alpha}$ attribution, and $\sigma \in \mathbb{T}(\vec{\alpha})$. Then for any $u \in \mathbb{U}$ with $u \neq \times$*

$$[[\sigma]^T_{\varphi,t}] = u \Rightarrow \begin{array}{l} t = u \text{ or} \\ \varphi(\alpha) = u \text{ for some } \alpha \text{ with } uniocc(\alpha, \sigma) \end{array}$$

PROOF. Easy □

6.6.14. PROPOSITION. *$\mathcal{E}_{\mathcal{A}}$ is uniqueness propagating.*

PROOF. Let $C \in \Sigma_{\mathcal{A}}$. Say $C\vec{\sigma}$ is the clause for C in the definition of $T\vec{\alpha}$. Let (φ, t) be a $T\vec{\alpha}$ attribution. If $\Sigma[\![\vec{\sigma}]\!]^T_{\varphi,t} = \times$ we are done. Suppose $\Sigma[\![\vec{\sigma}]\!]^T_{\varphi,t} = \triangle$. Say $[[\![\sigma_i]\!]^T_{\varphi,t}] = \triangle$. Then by Lemma 6.6.13 and the observation that for any j, $uniocc(\alpha_j, \sigma_i)$ implies $uniprop(T)_j$ one obtains $[[\![T\vec{\alpha}]\!]^T_{\varphi,t}] = t = \triangle$. The $\bullet$ case is treated similarly. □

Curry environments

First we describe how the types for each Curry variant F_i of F is obtained from the environment type for F. With respect to the underlying conventional types, the Currying operation for uniqueness types is essentially the same as in the conventional case. In addition, one has to keep track of uniqueness attributes while constructing the nested function types. If one of the arguments of a Curry variant is unique (i.e. $\bullet$ or $\triangle$) this uniqueness is *protected* by making the result of this partial application 'necessarily unique' ($\triangle$). This will prevent harmful copying of these applications; see the example below.

6.6.15. DEFINITION. (i) Let $u \in \mathbb{U}$ and $\vec{\sigma} \in \hat{\mathbb{T}}$. Then u is said to be *uniqueness fixating for $\vec{\sigma}$* if $u = \triangle$ whenever $\Sigma\vec{\sigma} \neq \times$; otherwise $u \in \{\bullet, \times\}$.

(ii) For each symbol type $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$ and $0 \leq j \leq k$, the set of *Curried versions of arity* j (notation $\sigma_j^{\mathcal{C}}$) is

$$\{(\sigma_1, \ldots, \sigma_j) \rightarrowtail \sigma_{j+1} \overset{u_j}{\rightarrow} \cdots \overset{u_{k-1}}{\rightarrow} \sigma_{k+1} \mid \text{each } u_i \text{ is uniqueness fixating for } (\sigma_1, \ldots, \sigma_i)\}.$$

Note that the function attributes are fixed as soon as one of the preceding σ_i is unique; in other cases there is some liberty. It will be convenient to isolate a *minimal* and a *maximal* result type of the j-th Curried version: the minimal variant is obtained by choosing $u_i = \bullet$ whenever possible; the maximal variant has $\times$ on such spots.

We consider type environments where each function symbol has exactly one type. This type determines the possible types for its Curry variants.

6.6.16. DEFINITION. (i) A *function type environment* (for $\mathfrak{T}$) is a map $\mathcal{F} : \Sigma_{\mathcal{F}}^{\mathrm{f}} \rightarrow \hat{\mathbb{T}}_S$.

(ii) Such a function type environment induces a uniqueness type environment for $\Sigma_{\mathcal{F}}^{\mathrm{f}}$ and $\Sigma_{\mathcal{F}}^{\mathrm{c}}$ (notation $\mathcal{E}_{\mathcal{F}}^{\mathrm{f}}$ and $\mathcal{E}_{\mathcal{F}}^{\mathrm{c}}$ respectively) in the following way.

$$\begin{aligned} \mathcal{E}_{\mathcal{F}}^{\mathrm{f}}(F) &= \{\mathcal{F}(F)\}, \\ \mathcal{E}_{\mathcal{F}}^{\mathrm{c}}(F_j) &= \mathcal{F}(F)_j^{\mathcal{C}}. \end{aligned}$$

(iii) $\mathcal{E}_{\mathcal{F}}$ denotes the combination of these environments for $\Sigma_{\mathcal{F}}$, i.e. $\mathcal{E}_{\mathcal{F}} = \mathcal{E}_{\mathcal{F}}^{\mathrm{f}} \cup \mathcal{E}_{\mathcal{F}}^{\mathrm{c}}$.

6.6.17. LEMMA. *Let $\mathcal{F}$ be a function environment.*

(i) *$\mathcal{E}_{\mathcal{F}}^{\mathrm{c}}$ is coercion consistent.*

(ii) *$\mathcal{E}_{\mathcal{F}}^{\mathrm{c}}$ is uniqueness propagating.*

PROOF. Easy. □

6.6.18. EXAMPLE. Let $\mathcal{F}(\mathbf{F}) = (\sigma^{\bullet}, \tau^{\times}) \rightarrowtail \rho^{\times}$. Then

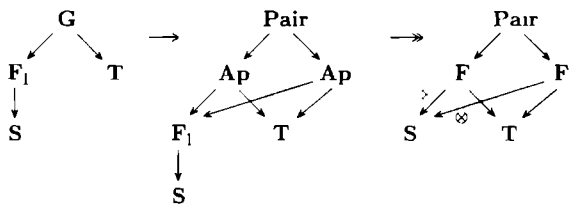
$$\begin{aligned} \mathcal{E}_{\mathcal{F}}(\mathbf{F}) &= \{(\sigma^{\bullet}, \tau^{\times}) \rightarrowtail \rho^{\times}\}, \\ \mathcal{E}_{\mathcal{F}}(\mathbf{F}_1) &= \{\sigma^{\bullet} \rightarrowtail \tau^{\times} \overset{\triangle}{\rightarrow} \rho^{\times}\}, \\ \mathcal{E}_{\mathcal{F}}(\mathbf{F}_0) &= \{\ \sigma^{\bullet} \overset{\bullet}{\rightarrow} \tau^{\times} \overset{\triangle}{\rightarrow} \rho^{\times}, \\ &\qquad \sigma^{\bullet} \overset{\times}{\rightarrow} \tau^{\times} \overset{\triangle}{\rightarrow} \rho^{\times}\ \}. \end{aligned}$$

The choice of the attribute $\triangle$ for the result type of $\mathbf{F}_1$ (instead of just $\bullet$) becomes clear if one realizes that uniqueness of the σ object may be destroyed if one allows the result type to be coerced to a $\times$-type. This is illustrated in the following.

6 6 19 NON-EXAMPLE Consider $\mathcal{F}$ with $\mathcal{F}(\mathbf{F})$ as in the above example, and $\mathcal{F}(\mathbf{G}) = (\tau^\times \xrightarrow{\times} \rho^\times, \tau^\times) \rightarrowtail \mathrm{Prod}^\bullet(\rho^\times, \rho^\times)$ Suppose one allows $\sigma^\bullet \rightarrowtail \tau^\times \xrightarrow{\bullet} \rho^\times$ as a type for $\mathbf{F}_1$ Then the rule

$$\mathbf{G}(f\ x) \to \mathbf{Pair}(\mathbf{Ap}(f,x), \mathbf{Ap}(f,x))$$

is typable taking **Pair** $(\alpha^\times, \beta^\times) \rightarrowtail \mathrm{Prod}^\times(\alpha^\times, \beta^\times)$, and using the fact that $\tau^\times \xrightarrow{\bullet} \rho^\times \leq^\otimes \tau^\times \xrightarrow{\times} \rho^\times$ Applying this rule to the first (well-typed) graph below (assuming $\mathbf{S}$ $\sigma^\bullet$ and $\mathbf{T}$ $\tau^\times$) leads to the second graph, and via two applicative reduction steps to the third, which is obviously not well-typed, since $\mathbf{S}$ is no longer unique



Combining environments

In the rest of this paper we will consider environments given in the following way

6 6 20 DEFINITION Let $\mathfrak{T}$ be an applicative TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment

(i) The *combined uniqueness environment* $\mathcal{E}_{\mathcal{F}\mathcal{A}}$ is obtained by setting

$$\mathcal{E}_{\mathcal{F}\mathcal{A}} = \mathcal{E}_0 \cup \mathcal{E}_{\mathcal{F}} \cup \mathcal{E}_{\mathcal{A}}$$

(ii) Typability in $\mathcal{F}, \mathcal{A}$ is denoted by writing $\mathcal{F}, \mathcal{A} \vdash g\ \ \sigma$ instead of $\mathcal{E}_{\mathcal{F},\mathcal{A}} \vdash g\ \sigma$

We allow Curry variants in functional rewrite rules only in a restricted way

6 6 21 DEFINITION Let $\mathcal{F}$ be a function environment for $\mathfrak{T}$ Then $\mathfrak{T}$ is said to be *Curry safe* for $\mathcal{F}$ if for any F_j occurring as pattern symbol in $\mathcal{R}_{\mathcal{F}}$ the following holds Say $|\mathcal{F}(F)| = (\sigma_1, \quad, \sigma_k) \rightarrowtail \tau$ For any $i \leq j$ and $\alpha \in \mathbb{V}$

$$occ(\alpha, \sigma_i) \sqsubseteq \ominus\ \ occ(\alpha, \sigma_{j+1}) \sqcup \quad \sqcup \ominus\ \ occ(\alpha, \sigma_k) \sqcup occ(\alpha, \tau)$$

Note that this is equivalent to '$\mathcal{E}^{\mathbf{c}}_{\mathcal{F}}$ is occurrence increasing w r t F_j'

6 6 22 ENVIRONMENT THEOREM *Let $\mathfrak{T}$ be a TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment for $\mathfrak{T}$ Suppose $\mathfrak{T}$ is Curry safe for $\mathcal{F}$ Then $\mathcal{E}_{\mathcal{F}\mathcal{A}}$ is uniqueness safe for $\mathfrak{T}$*

PROOF By Corollary 6 6 12, Proposition 6 6 14 and Lemma 6 6 17 □

6.7. Typing Redexes

In this section we describe the relation between typing and matching. If R is a functional rule that applies to g, then the uniqueness typing of the matching part in g is shown to be coercible to an instance of the R-typing. This has two consequences: the reduction result is typable with the corresponding instance of the right-hand side, and the uniqueness assumptions made in the left-hand side of R translate into locality properties of the matching nodes. The former is important in the proof of the subject reduction property (see Section 6.12); the latter shows that the typing system indeed guarantees the intended notion of 'uniqueness' (see Section 6.8).

The typing of a function symbol induces a typing notion for the corresponding Curry rules. This standard type construction will be described in the second part of this section. It will turn out that this typing also satisfies the above properties. Therefore, in the analysis performed in the rest of this paper, functional and applicative rewrite steps can be handled in a uniform way.

Fix a TGRS $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ over $\mathcal{A}$, and a function environment $\mathcal{F}$ for $\mathfrak{T}$. We assume that $\mathcal{R}$ is $\mathcal{F}, \mathcal{A}$-typable, and $\mathfrak{T}$ is Curry safe for $\mathcal{F}$.

Typing for functional redexes

We start with a general result.

6.7.1. LEMMA. *Let $\mu : g \xrightarrow[m]{} h$ be a match. Suppose $\mathcal{U}_g$ is a plain $\mathcal{E}$-uniqueness typing for g, and $\mathcal{U}_h$ is an $\mathcal{E}$-uniqueness typing for h according to use_h. Let $n \in g$. Suppose $\mathcal{E}$ is uniqueness safe w.r.t. $symb(n)$. Set $n_i = args_g(n)_i$.*

(i) *If $\mathcal{U}_h(\mu(n)) \leq \mathcal{U}_g(n)$, then $\mathcal{U}_h(\mu(n_i)) \leq \mathcal{U}_g(n_i)$.*

(ii) *If $[\mathcal{U}_g(n_i)] \neq \times$, then $[\mathcal{U}_g(n)] \neq \times$ and $use_h(\mu(n), i) = \odot$*

PROOF. Say $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(symb(n))$ such that, say, $\mathcal{U}_h(\mu(n)) = \tau^*$, $\mathcal{U}_g(n) = (\tau')^{*'}$ and $\mathcal{U}_h(\mu(n_i)) \leq^{use_h(\mu(n),i)} \sigma_i^*$, $\mathcal{U}_g(n_i) = (\sigma_i')^{*'}$. Then (i) follows since $\sigma_i^* \leq (\sigma_i')^{*'}$ by Proposition 6.6.5. If $[\mathcal{U}_g(n_i)] = [(\sigma_i')^{*'}] \neq \times$ then also $[\sigma_i^*] \neq \times$ by uniqueness propagation, and hence $use_h(\mu(n), i) = \odot$. This shows (ii). □

We now investigate typing according to $\mathcal{F}, \mathcal{A}$.

6.7.2. LEMMA. *Let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a redex in g with $R \in \mathcal{R}_{\mathcal{F}}$. Let $\mathcal{U}_g$ be a uniqueness typing for g according to use_g, and let $\mathcal{U}_R$ be a uniqueness typing for R. Set $l_i = args_R(l)_i$ for each appropriate i.*

(i) *There exists a substitution $*$ such that*

$$\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*,$$
$$\mathcal{U}_g(\mu(l_i)) \leq (\mathcal{U}_R(l_i))^*.$$

(ii) *If* $[\mathcal{U}_R(l_i)] \neq \times$ *then* $use_g(\mu(l), i) = \odot$.

PROOF. Say $\mathcal{F}(symb(n)) = \vec{\sigma} \rightarrowtail \tau$. Since $\mathcal{U}_g$ is a uniqueness typing one has

$$\mathcal{U}_g(\mu(l)) = \tau^*,$$
$$\mathcal{U}_g(\mu(l)_i) \leq^{use_g(\mu(l),i)} \sigma_i^*$$

for some $*$.

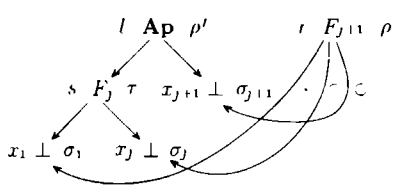
(i) Observe that $\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*$ by definition of rule typing. Moreover $\mathcal{U}_g(l_i) \leq (\mathcal{U}_R(l_i))^*$ since $\mathcal{U}_R$ is a plain uniqueness typing for $R \mid l$.

(ii) Suppose $[\mathcal{U}_R(l_i)] \neq \times$. Then also $[\sigma_i^*] \neq \times$ since $\mathcal{U}_R$ is a plain uniqueness typing. Hence $use_g(\mu(l), i) = \odot$. □

Typing for Curry redexes

The environment $\mathcal{F}$ induces the following typing notion for applicative redexes. Let $F \in \Sigma_{\mathcal{F}}$ with arity $k \geq 1$; say $\mathcal{F}(F) = \sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$.

The *standard uniqueness typing* for Ap_j^F (notation $\mathcal{U}_j^F$) is the type assignment indicated by the following picture. Here F_j, F_{j+1} are respectively the j-th and j+1-th Curry variant of F. Moreover, τ is the maximal result type of the j-th Curried version of σ, and ρ the minimal result type of the j+1-th Curried version Say $\tau = \sigma_{j+1} \xrightarrow{u} \rho'$.



6.7.3. LEMMA. $\mathcal{U}_j^F$ *is a uniqueness typing for* Ap_j^F *(according to the marking indicated above).*

PROOF. Obvious, since $\rho \leq \rho'$. □

6.7.4. LEMMA. *Let* $\Delta = \langle \mathrm{Ap}_j^F, \mu \rangle$ *be an (applicative) redex in* g *Suppose* $\mathcal{U}$ *is a uniqueness typing for* g, *according to* use_g. *Then one has the following.*

(i) *There exists a substitution* $*$ *such that*

$$\mathcal{U}(\mu(s)) \leq (\mathcal{U}_j^F(s))^*,$$
$$\mathcal{U}(\mu(x_i)) \leq (\mathcal{U}_j^F(x_i))^* \quad \textit{for all } i \leq j+1,$$
$$(\mathcal{U}_j^F(r))^* \leq \mathcal{U}(\mu(l)).$$

(ii) *For any node* $n = s, x_1, \ldots, x_{j+1}$

$$[\mathcal{U}_j^F(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}}) \text{ is not marked.}$$

PROOF. Since $\mathcal{U}$ is a uniqueness typing we have

$$\mathcal{U}(\mu(s)) = \sigma_{j+1}^* \xrightarrow{v} (\rho'')^*,$$
$$\mathcal{U}(\mu(x_i)) \leq^{use_g(\mu(s),i)} \sigma_i^* \text{ for all } i \leq j$$

for some substitution $*$, attribute v, and type ρ''. Note that $\rho'' \leq \rho'$ and $v \leq u$, since τ is maximal. Say $(\varphi \xrightarrow{w} \mathcal{U}(\mu(l)), \varphi) \rightarrowtail \mathcal{U}(\mu(l))$ is the used type instance of **Ap**. Then

$$\mathcal{U}(\mu(x_{j+1})) \leq^{use_g(\mu(l),2)} \varphi,$$
$$\sigma_{j+1}^* \xrightarrow{v} (\rho'')^* \leq \varphi \xrightarrow{w} \mathcal{U}(\mu(l)),$$
$$\mathcal{U}(\mu(l)) \leq (\rho'')^*.$$

As to (i), the result for $\mu(s)$ follows by maximality of τ. Moreover note that $\mathcal{U}(\mu(x_i)) \leq \sigma_i^*$ for any $i \leq j$, and $\mathcal{U}(\mu(x_{j+1})) \leq \varphi \leq \sigma_{j+1}^*$. The third result follows by minimality of ρ.

As to (ii), first note that $[\mathcal{U}_j^F(s)]\ (= [\tau]) = \triangle$ whenever it is unique. Therefore $use_g(\mu(l), i) = \odot$. For the x_i we distinguish two cases. Suppose $[\mathcal{U}_j^F(x_i)]\ (= \sigma_i) \neq \times$.

Case $i \leq j$. Since $\mathcal{U}(\mu(x_i)) \leq^{use_g(\mu(s),i)} \sigma_i^*$ one has $use_g(\mu(s), i) = \odot$. Moreover, observe that

$$[\sigma_i] \neq \times \;\Rightarrow\; [\tau] = \triangle.$$

Hence again $use_g(\mu(l), 1) = \odot$. Thus, $\mu(\overline{\overline{x_i}})$ is not marked.

Case $i = j + 1$. Now we can use that

$$\mathcal{U}(\mu(x_{j+1})) \leq^{use_g(\mu(l),2)} \varphi \leq \sigma_{j+1}^*.$$

Hence $use(\mu(l), 2) = \odot$. □

Matching and extension

6.7.5. MATCHING THEOREM. *Let* $g \in \mathcal{G}$, *and let* $\Delta = \langle R, \mu \rangle$ *be a redex in* g *with* $R \in \mathcal{R}$. *Let* $\mathcal{U}_g$ *be an* $\mathcal{F}, \mathcal{A}$*-uniqueness typing for* g *according to* use_g, *and let* $\mathcal{U}_R$ *be a* $\mathcal{F}, \mathcal{A}$*-typing for* R.

(i) *There exists a substitution* $*$ *such that*

$$(\mathcal{U}_R(\tau))^* \leq^{use(R)} \mathcal{U}_g(\mu(l)),$$
$$\mathcal{U}_g(\mu(n)) \leq (\mathcal{U}_R(n))^*, \text{ for any } n \in R \mid l \text{ with } n \neq l.$$

(ii) *For any* $n \in R \mid l$ *with* $n \neq l$

$$[\mathcal{U}_R(n)] \neq \times \Rightarrow \mu(\overline{n}) \text{ is not marked by } use_g$$

PROOF Distinguish cases as to the form of the rewrite rule

Case 1 R is functional Using Lemma 6 7 2 (i), determine $*$ such that $\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*$ and $\mathcal{U}_g(\mu(l_i)) \leq (\mathcal{U}_R(args_R(l)_i))^*$ As to (i), by substitutivity of the coercion relations (Lemma 6 4 16)

$$(\mathcal{U}_R(r))^* \leq^{use(R)} (\mathcal{U}_R(l))^* = \mathcal{U}_g(\mu(l))$$

The second property holds by Lemma 6 7 1 (i) (applied repeatedly) Moreover (ii) follows by repeated application of Lemma 6 7 1 (ii) and Lemma 6 7 2 (ii)

Case 2 R is applicative Then we are done by Lemma 6 7 4 □

6 7 6 COROLLARY (Extension Typing) *Let* $g \in \mathcal{G}$, *and let* $\Delta = \langle R, \mu \rangle$ *be a redex in* g *with* $R \in \mathcal{R}$ *Let* $\mathcal{U}_g$ *be an* $\mathcal{F}, \mathcal{A}$ *uniqueness typing for* g, *according to* use_g *Then there exists a marking use for* $R \mid r$ *and a uniqueness typing* $\mathcal{U}$ *for* $R \mid r$ *(according to use) such that*

(1) $\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l))$

and for any $n \in (R \mid l) \cap (R \mid r)$

(2) $\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n)$,

(3) $[\mathcal{U}(n)] \neq \times \Rightarrow \mu(\overline{n})$ *is not marked by* use_g

PROOF Let $\mathcal{U}_R$ be a uniqueness typing for R Set $use = use_R$ Set $\mathcal{U} = \mathcal{U}_R^* \restriction (R \mid r)$, where $*$ is obtained by the Matching Theorem Then (1), (2) and (3) hold Moreover $\mathcal{U}$ is a uniqueness typing for $R \mid r$ since the coercion relations are substitutive (Lemma 6 4 16) □

6.8. Locality Properties of Primary Redexes

The marking principle was introduced to analyze access dependencies in graphs The intention is that any actual argument in a primary redex is *local* for the root of the redex whenever its counterpart in the pattern of the rule is typed 'unique' This is proved by translating the uniqueness information in the rule pattern into a marking property of the path connecting the redex root and the argument in question

6 8 1 DEFINITION Let g be a graph, and $m, n \in g$

(i) n is *local for* m (in g) if either $m = n$ or

$$\forall p \quad r_g \rightsquigarrow n \; [m \in p]$$

(ii) n is *singly connected* to m if there exist exactly one path $p \quad m \rightsquigarrow n$

(iii) n is *unique* for m is n is local for m and n is singly connected to m.

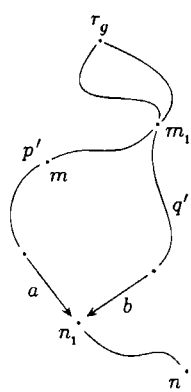
6.8.2. DEFINITION. Let p be a path in g. Then p is a *function-data path* if $\mathrm{r}(p)$ is a function node, and p^- is a data path.

6.8.3. LEMMA. *Let g be a graph, and let use be a marking function for g. Let $p : m \rightsquigarrow n$ be a function-data path. Suppose m is a primary node. If p is not marked by use then one has the following.*

(i) *n is local for m in g.*

(ii) *If m is a primary node then n is singly connected to m.*

PROOF. (i) If $m = n$ then we are done immediately, so assume $m \neq n$. Say $p_0 : r_g \rightsquigarrow m$ be an acyclic primary path. Let $q : r_g \rightsquigarrow n$. We have to show that $m \in q$. We can assume that q is acyclic. Suppose, towards a contradiction, that $m \notin q$. Then there exists a node n_1 on $\tilde{p}$ and $a, b \in acc(n_1)$ such that $a \neq b$ and $a \in p$, $b \in q$. Now there are subpaths p' of $p_0 * p$ and q' of q such that $(p', a) \wedge (q', b)$ is a critical path combination, say with joining node m_1.



We claim that $p' * a \lesssim q' * b$. Then $p' * a$ is marked, contradicting the assumption that p is not marked

Proof of the claim. Consider the position of m_1 on $p_0 * p$.

Case 1. $m_1 \in p$. Note that $m_1 \neq m$ by assumption Then m_1 is a data node, so $p' * a \sim q' * b$.

Case 2. $m_1 \in p_0$. Then $p' * a$ starts with a primary reference, so $p' * a \lesssim q' * b$.

(ii) Let $q : m \rightsquigarrow n$. Suppose, towards a contradiction, that $q \neq p$. Say a the first reference on q such that $a \notin p$. Note that $a \lesssim b$. Then a and b are the top references of some critical path combination causing a mark on p. Contradiction. □

6.8.4. UNIQUENESS THEOREM. *Let $\mathfrak{T}$ be Curry safe for $\mathcal{F}$. Let g be a $\mathcal{F},\mathcal{A}$-typable graph. Let $\Delta = (R,\mu)$ be a primary redex in g with $\mathcal{F},\mathcal{A}$-uniqueness typing $\mathcal{U}$. Let $n \in R \mid l$, $n \neq l$ be a primary node. Then*

$$[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(n) \text{ is unique for } \mu(l) \text{ in } g.$$

PROOF. Say use_g is an appropriate marking function. By the Matching Theorem 6.7.5, the function-data path $\mu(\overline{n})$ is not marked by use_g. Now the result follows by Lemma 6.8.3. □

Now the compile time garbage collection (mentioned in the Introduction) can be made explicit. If a (primary) left-hand side node is typed 'unique' and it is not re-used in the right-hand side, then the corresponding node in the object graph will certainly become garbage.

6.9. Saturated Markings

In the sections 6.10, 6.11 and 6.12, it will be shown that uniqueness typability is preserved during reduction. This splits into two parts. Firstly, if $g \xrightarrow{A} h$ one has to prove that the markings use_g and use_R (of g and the applied rewrite rule respectively) can be combined into a proper marking of h. Furthermore it needs to be verified that this marking admits a suitable uniqueness typing.

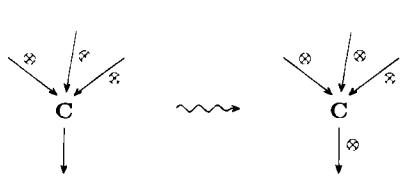
It will be convenient to transform a given marking function for g into a *saturated* one, i.e. a marking that exhibits a 'maximal' labelling. This transformation is based on the following observation. If all paths (from r_g) to a certain data node are marked, then each direct argument of this node cannot be used destructively by any function accessing this argument via the data node in question. In other words: access via this data node to such a direct argument will necessarily be considered 'read-only'. The corresponding references can therefore safely be marked with $\otimes$.

6.9.1. DEFINITION. A marking *use* for g is *saturated* if for any data node $n \in g$ the following holds. If every path $p : r_g \rightsquigarrow n$ is marked, then for all $\imath \leq \text{arity}(symb_g(n))$

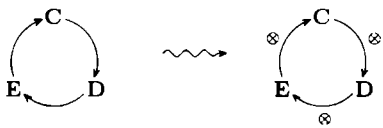
$$use((n,\imath)) = \otimes.$$

In this section we will present a method for transforming a marking function into a saturated one, in such a way that typability is maintained. This will be done via two operations.

The first operation is $\otimes$-*extension* for data nodes to which all references are labelled $\otimes$.



Secondly, constructor cycles can be marked completely.



We first prove some type theoretical results. For certain uniqueness types σ one can construct a 'non-unique variant' $\langle\sigma\rangle$ such that $\sigma \leq \langle\sigma\rangle$.

6.9.2. DEFINITION. (i) The *uniqueness removal map* $\langle\ \rangle : \widehat{\mathbb{T}} \to \mathbb{T}^{\times} \cup \{\uparrow\}$ (where $\uparrow$ represents 'undefined') is specified inductively as follows. (Here $\to$ is again regarded as an ordinary constructor.)

$$\begin{aligned}
\langle\sigma\rangle &= \sigma \quad \text{if } [\sigma] = \times,\\
\langle\sigma\rangle &= \uparrow \quad \text{if } [\sigma] = \triangle,\\
\langle\alpha^{\bullet}\rangle &= \uparrow,\\
\langle T^{\bullet}(\sigma_1,\ldots,\sigma_k)\rangle &= T^{\times}(\sigma'_1,\ldots,\sigma'_k),\\
&\quad \text{where } \sigma'_i = \langle\sigma_i\rangle \text{ if } \mathit{uniprop}(T)_i,\\
&\qquad\qquad = \sigma_i \quad \text{otherwise.}
\end{aligned}$$

It is understood that $\langle T^{\bullet}(\sigma_1,\ldots,\sigma_k)\rangle$ is $\uparrow$ whenever one of the σ'_i equals $\uparrow$.

(ii) We will write $\langle\sigma\rangle\!\downarrow$ to indicate that $\langle\sigma\rangle \neq \uparrow$.

6.9.3. LEMMA. $\langle\ \rangle$ *is well defined, i.e. for all* $\sigma \in \widehat{\mathbb{T}}$ *one has* $\langle\sigma\rangle \in \widehat{\mathbb{T}}$ *whenever* $\langle\sigma\rangle\!\downarrow$.

PROOF. One easily checks that $\langle\sigma\rangle$ satisfies the admissibility requirements for type attributes. □

6.9.4. LEMMA. (i) σ *is* $\leq^{\otimes}$*-coercible* $\Rightarrow$ $\langle\sigma\rangle\!\downarrow$.

(ii) $\sigma \leq^{\otimes} \tau \Rightarrow \sigma \leq \langle\sigma\rangle \leq^{\otimes} \tau$.

PROOF. Since the arguments are similar we will prove the statements (i) and (ii) simultaneously by induction on the structure of σ. First observe that $[\sigma] \neq \triangle$. If $[\sigma] = \times$ then we are done. Furthermore, $\sigma = \alpha^{\bullet}$ is impossible. Now suppose $\sigma = T^{\bullet}(\sigma_1,\ldots,\sigma_k)$. Then $\tau = T^{\times}(\tau_1,\ldots,\tau_k)$ for some $\tau_1,\ldots,\tau_k$ with $\vec{\sigma} \overset{sign(T)}{\leq} \vec{\tau}$. As in Definition 6.9.2, write $\langle\sigma\rangle = T^{\times}\vec{\sigma'}$. We claim that $\sigma'_i\!\downarrow$ and $\sigma_i \leq \sigma'_i \leq \tau_i$ for all $i \leq k$. Note that this implies the desired result. Indeed, if not $\mathit{uniprop}(T)_i$ then $\sigma'_i = \sigma_i$ so we are done. Otherwise $[\tau_i] = \times$ and $\sigma'_i = \langle\sigma_i\rangle$. Moreover, $\mathit{uniprop}(T)_i$ implies $\mathit{sign}(T)_i \sqsupseteq \oplus$. In case $\mathit{sign}(T)_i = \top$ one has $\sigma_i \overset{\top}{\leq} \tau_i$. Hence $[\sigma_i] = \times$ so again $\sigma'_i = \sigma_i$. The remaining case is $\mathit{sign}(T)_i = \oplus$. Since $[\tau_i] = \times$ one has $\sigma_i \leq^{\otimes} \tau_i$. Now the induction hypothesis applies, so both $\sigma'_i\!\downarrow$ and $\sigma_i \leq \sigma'_i \leq \tau_i$ for all $i \leq k$. □

6.9.5. PROPOSITION. *Let C be a data symbol. Let $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(C)$. Let $*$ be a substitution for this type. Suppose τ^* is $\leq^{\otimes}$-coercible. Then there exists $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(C)$ and an appropriate substitution $*'$ such that*

$$\tau^* \leq (\tau')^{*'}$$

and for each i

$$\sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$$

and moreover for each ρ with $\tau^* \leq^{\otimes} \rho$

$$(\tau')^{*'} \leq^{\otimes} \rho.$$

PROOF. Note that $\langle \tau^* \rangle \downarrow$ by Lemma 6.9.4 (i).

Case 1. C is an algebraic constructor. Then $\langle \tau^* \rangle = (\tau')^{*'}$ with $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{A}}(C)$ for some $\vec{\sigma'}$, since $\mathcal{E}_{\mathcal{A}}(C)$ contains all $[\![\ ]\!]^{\tau}_{\varphi,t}$-variants of constructor types for admissible φ, t Observe that $\tau^* \leq (\tau')^{*'}$ by Lemma 6.9.4 (ii). Moreover $\sigma_i^* \leq (\sigma_i')^{*'}$ by Proposition 6.6.5. Note that $[(\tau')^{*'}] = \times$. Hence by uniqueness propagation of $\mathcal{E}_{\mathcal{A}}$ (Proposition 6.6.14) one has $[(\sigma_i')^{*'}] = \times$ for each i, so $\sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$. The third property follows from Lemma 6.9.4 (iii).

Case 2. C is a Curry variant. Then τ is of the form $\tau_1 \stackrel{u}{\rightarrow} \tau_2$. Note that $\langle \tau \rangle = \tau_1 \stackrel{\times}{\rightarrow} \tau_2$ since neither of the $\rightarrow$ positions is uniqueness propagating. Moreover $\langle \tau \rangle$ is a valid result type for the C-variant. Hence one can take $\vec{\sigma'} = \vec{\sigma}$, $\tau' = \langle \tau \rangle$, and $*' = *$. □

NOTATION. (i) If f is a function, then $f[x \mapsto p]$ denotes the function f' such that

$$\begin{aligned} f'(x) &= p, \\ f'(y) &= f(y) \qquad \text{if } y \neq x. \end{aligned}$$

Multiple assignments of the form $f[x \mapsto p, y \mapsto q]$ are also used.

(ii) The ordering $\leq$ on uniqueness typings for a graph g is interpreted pointwise, by

$$\mathcal{U} \leq \mathcal{U}' \ \Leftrightarrow\ \forall n \in g\ \ [\mathcal{U}(n) \leq \mathcal{U}'(n)].$$

The following gives a justification for the $\otimes$-extension operation.

6.9.6. PROPOSITION. *Let $\mathcal{U}$ be a $\mathcal{F},\mathcal{A}$-uniqueness typing for g according to use. Let $n \in g$, $n \neq r_g$ be a data node (say with arity k) such that $use(a) = \otimes$ for all $a \in acc(n)$. Set $use' = use[(n,1) \mapsto \otimes, \ldots, (n,k) \mapsto \otimes]$. Then there exists a $\mathcal{F},\mathcal{A}$-uniqueness typing $\mathcal{U}'$ according to use' such that $\mathcal{U} \leq \mathcal{U}'$ and $\mathcal{U}'(r_g) = \mathcal{U}(r_g)$.*

PROOF. Set $n_i = args(n)_i$. Say $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(symb(n))$, such that

$$\begin{aligned} \mathcal{U}(n) &= \tau^*, \\ \mathcal{U}(n_i) &\leq^{use(n,i)} \sigma_i^* \qquad \text{for all } i. \end{aligned}$$

By Proposition 6.9.5 there exists $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(symb(n))$ and a substitution $*'$ such that

$$\begin{aligned} \tau^* &\leq (\tau')^{*'}, \\ \sigma_i^* &\leq^{\otimes} (\sigma_i')^{*'} \qquad \text{for all } i. \end{aligned}$$

Now take $\mathcal{U}' = \mathcal{U}[n \mapsto (\tau')^{*'}]$. Then

$$\mathcal{U}'(n_i) \leq^{use(n,i)} \sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$$

so by transitivity of $\leq$

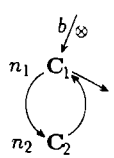
$$\mathcal{U}'(n_i) \leq^{\otimes} (\sigma_i')^{*'}$$

and we are done. □

The following justifies the cycle-markings described above.

6.9.7. PROPOSITION. *Let $\mathcal{U}$ be a $\mathcal{F},\mathcal{A}$-uniqueness typing for g, according to use. Let $p : n \rightsquigarrow n$ be a data cycle; write $p = (a_1, \ldots, a_n)$. Set $use' = use[a_1 \mapsto \otimes, \ldots, a_n \mapsto \otimes]$. Then there exists a $\mathcal{F},\mathcal{A}$-uniqueness typing $\mathcal{U}'$ according to use' such that $\mathcal{U} \leq \mathcal{U}'$ and $\mathcal{U}'(r_g) = \mathcal{U}(r_g)$.*

PROOF. To avoid tedious denotational matters, we treat an example with two data nodes. From this the general idea will become clear. Consider the following situation.



There exists an external reference b to this cycle, say to n_1. Since $use(b) = \otimes$ the type $\mathcal{U}(n_1)$ is $\leq^{\otimes}$-coercible. Say in $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ one has

$$\begin{aligned} \mathbf{C}_1 \qquad & (\sigma, \tau) \rightarrowtail \rho, \\ \mathbf{C}_2 \qquad & \chi \rightarrowtail \psi, \end{aligned}$$

such that $\mathcal{U}(n_1) = \rho^{*_1}$, $\mathcal{U}(n_2) = \psi^{*_2}$ and $\psi^{*_2} \leq \sigma^{*_1}$, $\rho^{*_1} \leq \chi^{*_2}$ respectively. By Proposition 6.9.5 there exists an environment type $(\sigma', \tau') \rightarrowtail \rho'$ for $\mathbf{C}_1$ and a substitution $*_1'$ such that

$$\begin{aligned} \rho^{*_1} &\leq (\rho')^{*_1'}, \\ \sigma^{*_1} &\leq^{\otimes} (\sigma')^{*_1'}. \end{aligned}$$

Hence ψ^{*_2} ($\leq \sigma^{*_1}$) is $\leq^{\otimes}$-coercible so again by Proposition 6 9 5 is an environment type $\chi' \rightarrowtail \psi'$ for $\mathbf{C}_2$ and a substitution $*_2'$ such that

$$\begin{aligned} \psi^{*_2} &\leq (\psi')^{*_2'}, \\ \chi^{*_2} &\leq^{\otimes} (\chi')^{*_2'} \end{aligned}$$

Define $\mathcal{U}' = \mathcal{U}[n_1 \mapsto (\rho')^{*_1}, n_2 \mapsto (\chi')^{*_2'}]$ It remains to show that $\mathcal{U}'$ is a uniqueness typing according to *use'* Since the references to n_1, n_2 not occurring on p are $\otimes$-labelled, coercions along these references remain valid by Proposition 6 9 5 In order to show that $\mathcal{U}'(n_2) \leq^{\otimes} (\sigma')^{*_1'}$, it is sufficient to prove that $\psi^{*_2} \leq^{\otimes} (\sigma')^{*_1'}$ Indeed,

$$\begin{aligned} \psi^{*_2} &\leq \sigma^{*_1} \\ &\leq^{\otimes} (\sigma')^{*_1'} \end{aligned}$$

and we are done Similarly one shows $\mathcal{U}'(n_1) \leq^{\otimes} (\chi')^{*_2}$ □

This provides a method for constructing saturated marking functions

6 9 8 DEFINITION Let *use* be a marking function for g Then use^+ is the marking function obtained by adjusting *use* in the following ways

(1) Each data cycle is completely marked

(2) $\otimes$ extension is performed repeatedly while possible

6 9 9 THEOREM *Let use be a marking function for g Then* use^+ *is saturated*

PROOF Suppose n is a data node in g such that every path $p \quad r_g \rightsquigarrow n$ is marked We will show that $use^+(a) = \otimes$ for any $a \in acc(n)$ Then we are done since use^+ is closed under $\otimes$-extension

It will be shown that there exists no non-empty unmarked acyclic path to n Note that this implies the above statement Suppose, towards a contradiction, q is such a path of maximal length Write $q = (m, \imath) * q'$ with q' a data path Note that m is not a function node (otherwise there is an unmarked path from r to n)

Claim $use^+(a) = \otimes$ for all $a \in acc(m)$ This contradicts the assumption that q is not marked

Proof of the claim Let $a = (m', \jmath) \in acc(m)$ If m' is a function node then $use^+(a) = \otimes$ since every path from r to n is marked Suppose m' is a data node If $m' \in q$ then $use^+(a) = \otimes$ since each data cycle is completely marked by use^+ If, on the other hand, $a * q$ is acyclic then $use^+(a) = \otimes$ by maximality of q This proves the claim □

6 9 10 SATURATION THEOREM Let g be a graph Suppose $\mathcal{F}, \mathcal{A} \vdash_{use} g \quad \sigma$ Then there exists a marking function *use'* such that

(1) *use'* is saturated,

(2) $\mathcal{F}, \mathcal{A} \vdash_{use'} g \quad \sigma$

PROOF By Theorem 6 9 9 and the propositions 6 9 6 and 6 9 7 □

6.10. Marking Reducts

In this section we will give a method for constructing a marking function for a reduction result, based on a (saturated) marking for the object graph and a marking for the right-hand side of the applied rewrite rule. We distinguish to cases as to the form of the rewrite rule. Note that application rules are extending.

Markings for extending reductions

In the sequel, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be an extending $\mathcal{R}$-redex in g. Say $g \overset{\Delta}{\to} h$.

Before making this precise we introduce some terminology.

6.10.1. DEFINITION. (i) A reference $a \in R|r$ is called a *border reference* if $a \notin R|l$ and $\mathrm{d}(a) \in R \mid l$. The collection of border references in R is indicated by B_R.

(ii) Let $a \in B_R$. Then $\overline{\overline{a}}$ denotes the path $\overline{\overline{\mathrm{d}(a)}}$ in the tree $R \mid l$. Similarly we use $\overline{a}$.

6.10.2. DEFINITION. Let a be a reference in h.

(i) a is called *new* if a is a reference in $R \mid r$ but not in $R \mid l$. Note that the starting node of a is a new node in h. Other references are called *old*. This terminology carries over to paths: a path is *new* if it contains a new reference; otherwise it is *old*.

(ii) a is *redirected* if $\mathrm{d}_g(a) = \mu(l)$ (and consequently $\mathrm{d}_h(a) = r_R$).

First we describe how a marking for h can be obtained from the respective markings for g and R. Since the part of g matching the left-hand side of R may contain more sharing than $R \mid l$, one could expect that the marking of $R \mid r$ might be too liberal: it may contain too few $\otimes$-labelled references in order to result in a proper marking for the reduct. However, arbitrary changing $\odot$-labels into $\otimes$ reduces coercion possibilities. It will turn out to be safe to add a $\otimes$ mark to any border reference a for which $\mu(\overline{\overline{a}})$ is marked

For convenience, we introduce some auxiliary operations concerning marking functions.

6.10.3. DEFINITION. (i) Let p be a path in a graph g, and *use* a marking for g. Then

$$\begin{aligned} \mathit{use}(p) &= \otimes && \text{if } p \text{ is marked,} \\ &= \odot && \text{otherwise.} \end{aligned}$$

(ii) The operation $+$ on $\mathbb{M}$ is defined by

$$\begin{aligned} u + v &= \otimes && \text{if } u = \otimes \text{ or } v = \otimes, \\ &= \odot && \text{otherwise.} \end{aligned}$$

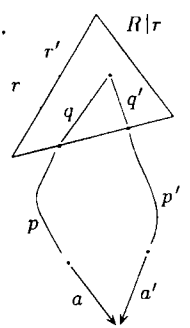
The following describes the construction of a marking for h based on markings for g and R.

6.10.4. DEFINITION. Let *use* be a marking for g, and use_R the rule marking of R. The *combined use function* use^Δ on Ref_h is defined as follows.

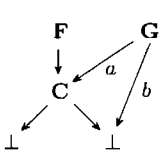
$$\begin{array}{lcll} use^\Delta(a) & = & use(a) & \text{if } a \text{ is old and } a \text{ is not redirected,} \\ & = & use(a) + use_R(R) & \text{if } a \text{ is redirected,} \\ & = & use_R(a) & \text{if } a \text{ is new and } a \text{ is not a border reference,} \\ & = & use_R(a) + use(\mu(\overline{a})) & \text{if } a \text{ is a border reference.} \end{array}$$

Note that two corrections are made on *use* and use_R: a border reference a is marked if either it is marked in R or $\mu(\overline{a})$ is marked in g. Moreover redirected references are corrected according to the root attribute of $R \mid r$.

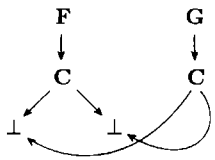
As said before, a marking function for $R \mid r$ might be an insufficient marking for the extension part (containing the new nodes) of h, e.g. in the case of sharing (in h) 'below' $R \mid r$; see the following picture.



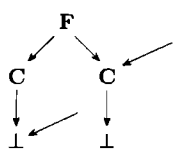
Suppose this introduces a critical path combination in which one of the paths needs to be marked (e.g. $q \lesssim q'$), and neither *use* nor use_R gives a $\otimes$-label on this path. Now consider the paths $r * p * a$ and $r' * p' * a'$ both starting in $\mu(R \mid l)$. If r and r' do not coincide then consistency of R w.r.t. the argument classification implies that r is marked in g. Consequently, this marking is copied onto $R \mid r$ by the correction of border references. However, if r and r' coincide this does not work. This occurs if border references appear in a configuration of the following form.



Here a and b are said to be *stacked*: $\bar{\bar{a}}$ is an initial part of $\bar{\bar{b}}$. We do not allow such stacking of border references in rewrite rules. Note that this does not really reduce the expressive power of graph rewriting: one could replace the above rule by the following.



Unfortunately, there are pathological cases in which this restriction is insufficient. This will appear in a special case in the correctness proof. The problem is caused by pairs of border references which point into two distinct but compatible subpatterns of R (let us call them *target patterns*), in such a way that they are 'stacked modulo compatibility', as shown in the following figure.



(Applying this rule to a graph of the form

results in the creation of new references that are stacked.) Call the index of the reference from l to a subpattern the *index* of that subpattern. Say P is one of the target patterns in question. If this stack suspect situation appears, then it is necessary that the index of *any* P-compatible subpattern to be $\lesssim$-related with the index of P. In order to make this precise, some terminology is introduced.

6.10.5. DEFINITION. Let g be a graph.

(i) Let $p = ((n_1, i_1), \ldots, (n_\ell, i_\ell))$ be a path in g. The *route* of p is the sequence $(i_1, i_2, \ldots, i_\ell)$.

(ii) If g is a tree, then *address* of n (in g) is the route of the path from r_g to n.

6.10.6. DEFINITION. Let R be a rewrite rule of arity k.

(i) Let $a \in B_R$. We use the denotation $\hat{a}$ to indicate the address of the node $\mathrm{d}(a)$ (in $R \mid l$). Note that $(\hat{a})_1$ is the index of the subpattern p containing $\mathrm{d}(a)$ whereas $(\hat{a})^-$ is the address of $\mathrm{d}(a)$ in p.

(ii) Let $i, j \leq k$. Then i and j are *pattern overlapping* (notation $i \uparrow j$) if

$$(R \mid args(l)_i) \uparrow (R \mid args(l)_j).$$

(iii) Let $a, b \in B_R$. Then a is *potentially stacked on* b if

$$(\hat{a})_1 \uparrow (\hat{b})_1 \text{ and } (\hat{a})^- \subset (\hat{b})^-.$$

If a is potentially stacked on b then both a and b are called *stack suspect*.

(iv) R is *stack safe* if for each stack suspect $a \in B_R$ one has the following. Set $j = (\hat{a})_1$.

$$\forall i \leq k\, [i \uparrow j \;\Rightarrow\; i \lesssim j].$$

In particular rewrite rules for simple function symbols are stack safe.

6.10.7. LEMMA *Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Suppose R is stack safe. Moreover let a be stack suspect; say $(\hat{a})_1 = j$. If $\mu(args(l)_i) = \mu(args(l)_j)$, then $i \lesssim j$.*

PROOF. Set $a_i = args(l)_i$ and $a_j = args(l)_j$. Observe that $\mu : R \mid a_i \underset{\mathrm{rml}}{\rightrightarrows} g \mid \mu(a_i)$ and $\mu : R \mid a_j \underset{\mathrm{rml}}{\rightrightarrows} g \mid \mu(a_j)$. So $i \uparrow j$ and hence by stack safety one has $i \lesssim j$. □

In the sequel, we assume that any $R \in \mathcal{R}$ is stack safe.

6.10.8. THEOREM. *Let use_g be a saturated marking for g, and use_R a marking for R. Then the labelling use_g^{Δ} is a marking for h.*

PROOF. See Section 6.11. □

Markings for projections

For this subsection, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a projecting $\mathcal{R}$-redex in g. Say $g \overset{\Delta}{\to} h$.

6.10.9. DEFINITION. Let a be a reference in h. Then a is called *new* if $\mathrm{d}_g(a) = \mu(l)$, and consequently $\mathrm{d}_h(a) = \mu(r)$. Otherwise a is called *old*.

Now we are ready to describe the construction of a marking function for h, based on a marking for g.

6.10.10. DEFINITION. Let *use* be a marking for g. The *redirection marking* use^{Δ} on Ref_h is defined by

$$\begin{aligned} use^{\Delta}(a) &= use(a) && \text{if } a \text{ is old,} \\ &= use(a) + use_g(\mu(\overline{r})) && \text{if } a \text{ is new.} \end{aligned}$$

In this case only one correction on *use* is made, depending on the presence of a mark on $\mu(\bar{r})$.

6.10.11. THEOREM. *Let use_g be a saturated marking for g. Then the labelling use_g^{Δ} is a marking for h.*

PROOF. See Section 6.11. □

6.11. Correctness of Reduct Markings

We suggest that the reader skips the (very technical) correctness proofs at first reading. The complexity is mainly caused by the presence of cycles.

In the sequel, let g be a graph, and let *use* be a marking function for g. We will first prove some technical results concerning marking functions. The main part involves an analysis of the relative positions of critical path combinations and a treatment of cycles. As a warming-up, we start with some trivialities.

6.11.1. REMARK. Let $p = p_1 * p_2$ be a path.

(i) If p_2 is marked then p is marked.

(ii) Let p_1' be a marked path which is extendible with p_2. If p is marked then $p_1' * p_2$ is marked.

(iii) If p_1 is marked and p_2 is a data path then p is marked.

Some terminology concerning paths is necessary.

6.11.2. DEFINITION. Let p, q be paths.

(i) p and q *form a diamond* (notation $p \diamond q$) if p and q both diverge and converge, i.e. either $p = ()$ or $q = ()$ or

$$\mathrm{r}(p) = \mathrm{r}(q) \text{ and } \mathrm{d}(p) = \mathrm{d}(q) \text{ and } (\tilde{p})^{-}_{-} \# (\tilde{q})^{-}_{-}.$$

We use $p \diamondsuit q$ to indicate that the diamond is non-trivial, i.e. $p \diamond q$ and at least one of the paths p, q is non-empty.

(ii) p and q are *parallel* (notation $p \parallel q$) if $\tilde{p} = \tilde{q}$.

6.11.3. DIAMOND FACTORIZATION. Let $p, q : n \rightsquigarrow m$ be paths in g such that $p \neq q$. Then there exist paths p_0, q_0, p_1, q_1, r with $p_1 \diamondsuit q_1$ such that

$$\begin{aligned} p &= p_0 * p_1 * r, \\ q &= q_0 * q_1 * r. \end{aligned}$$

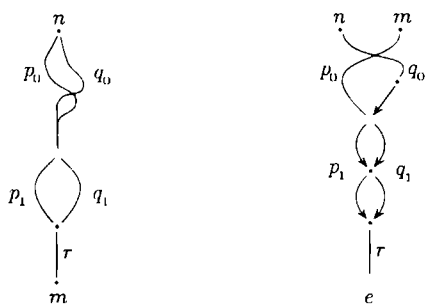
We say that the above paths p_0, q_0, p_1, q_1, r are a *diamond factorization* of p, q.

6 11 4 TAIL FACTORIZATION Let $p \quad n \rightsquigarrow e$ and $q \quad m \rightsquigarrow e$ Then there exist paths p_0, q_0, p_1, q_1, r, with $p_1 \parallel q_1$ and

$$\begin{aligned} p &= p_0 * p_1 * r, \\ q &= q_0 * q_1 * r \end{aligned}$$

We say that the above paths p_0, q_0, p_1, q_1, r are the (unique) *tail factorization* of p, q if r and $p_1 * r$ (and hence also $q_1 * r$) are of maximal length Observe that $q_0 * q_1$ is non empty if q is not a tail part of p Furthermore, note that q_0 is non-empty if $m \notin p$ and $m \neq e$ We call the last reference of q_0 the *entrance reference* of q to p

The above factorizations can be visualized as follows



6 11 5 PROPOSITION *Let $p, q \quad n \rightsquigarrow m$, $p \neq q$ be paths such that p and q intersect (apart from n, m) only in data nodes Furthermore, suppose p, q are not root cyclic Then*

$$p \lesssim q \quad \Rightarrow \quad p \text{ is marked}$$

PROOF Let p_0, q_0, p_1, q_1, r be a diamond factorization of p, q Then $p_1 \diamond q_1$ Say n_0 be the first node of p_1 (and of q_1)

Case 1 p_1 and q_1 are not empty Observe that r is a data path

Case 1a $n_0 = n$ Then p_0, q_0 are empty (otherwise p or q would have been root cyclic) So by assumption $p_1 \lesssim q_1$, and hence p_1 is marked By Remark 6 11 1 (ii) also $p_1 * r$ is marked

Case 1b $n_0 \neq n$ Since n_0 is a data node, it is simple, and again $p_1 \lesssim q_1$ Consequently, p_1 and therefore $p_0 * p_1 * r$ are marked

Case 2 p_1 is empty, and q_1 is non-empty Note that $q_1 \quad n_0 \rightsquigarrow n_0$ Since q is not root cyclic it follows that $n \neq n_0$ So p_0 and q_0 are not empty Say b is the last reference of p_0, i e $p_0 = p_0' * b$ Then one has $b \sim b * q_1$ which implies that b is marked Therefore also $p_0' * b * r = p$ is marked

Case 3 q_1 is empty, and p_1 not Similar to case 2 □

6.11.6. LEMMA. *Let $p*a : n \rightsquigarrow n$ be a cycle. Furthermore, let $b \in acc(n)$. Suppose $\mathrm{r}(b) \notin p$. Then one has the following.*

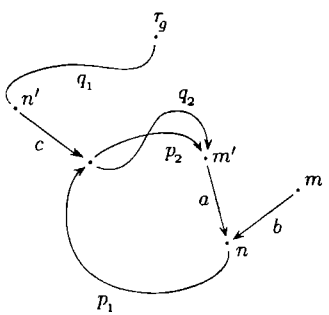
(i) *b and $b*p*a$ are marked.*

(ii) *If use is saturated then a is marked.*

PROOF. Say $m = \mathrm{r}(b)$.

(i) Set $q = b*p*a$. Observe that $b, q : m \rightsquigarrow n$ and that $b \sim q$. Furthermore, q is not root cyclic. Then b and q are marked by Proposition 6.11.5.

(ii) Say $m' = \mathrm{r}(a)$. If m' is a function node we are done, since $b*p*a$ is marked. Assume m' is a data node. Moreover suppose, towards a contradiction, that $use(a) \neq \otimes$. We will derive that every path leading to m' is marked. This contradicts the assumption, by saturation. Indeed, let $q : r_g \rightsquigarrow m'$. Say $c = (n', j)$ is the entrance reference of q to p. Write $q = q_1 * c * q_2$.



By (i) one has $use(c) = \otimes$. Write $p = p_1 * p_2$ with $p_1 : n \rightsquigarrow \mathrm{d}(c)$ and $p_2 : \mathrm{d}(c) \rightsquigarrow m'$. Now consider that path $p_1 * q_2 * a$ and the reference b. Again by (i) one has $b * p_1 * q_2 * a$ is marked, and hence $c * q_2 * a$ is marked (by Remark 6.11.1 (ii)). By assumption a is not marked so $c * q_2$ is marked. □

6.11.7. PROPOSITION. *Let $p * a : n \rightsquigarrow n$, and let $b \in acc(n)$. If $\mathrm{r}(a) \neq \mathrm{r}(b)$ then $use(a) = use(b) = \otimes$.*

PROOF. Say $n_a = \mathrm{r}(a)$ and $n_b = \mathrm{r}(b)$. By course-of-values induction on the length of p. Suppose $|p| = k$, and the statement holds for paths of smaller length. If $n_b \notin p$ then we are done by Lemma 6.11.6 (this covers in particular the case $|p| = 0$). Assume $n_b \in p$. Write $p = p_1 * p_2$ with $p_1 : n \rightsquigarrow n_b$ and $p_2 : n_b \rightsquigarrow n_a$. Since $n_a \neq n_b$, p_2 is not empty and hence $|p_1| < k$. Consider the path $p_1 * b$ and the reference a. By induction hypothesis $use(a) = use(b) = \otimes$. □

6.11.8. COROLLARY. *Let $p * a : n \rightsquigarrow n$, and let $b \in acc(n)$. Suppose $a \neq b$. If $\mathrm{r}(a) \neq \mathrm{r}(b)$ or $\mathrm{r}(a), \mathrm{r}(b)$ are data nodes then $use(a) = use(b) = \otimes$.*

6.11.9. PROPOSITION. *Let p be a cycle, say $p : n \rightsquigarrow n$, and let $q : m \rightsquigarrow n$ such that $p \neq q$. Suppose p_1, q_1, p_2, q_2, r is the tail factorization of p, q. Furthermore, suppose r is a data path, and q_2 (and hence also p_2) is a simple path. If $q_1 * q_2$ is not empty then p and q are marked.*

PROOF. By a case distinction.

Case 1. q_2 is not empty. Write $q_2 = q_2' * a$ and $p_2 = p_2' * b$. Since a and b start with the same simple node both references are marked, showing that p and q are marked.

Case 2. q_2 is empty and q_1 is not empty. Suppose p_1 is not empty. Write $q_1 = q_1' * a$ and $p_1 = p_1' * b$. Observe that a and b start with different nodes. By Lemma 6.11.6 $use(a) = use(b) = \otimes$, and hence p and q are marked. Now suppose p_1 is empty, and hence $p = r$ and $q = q_1 * r$. Say c is the entrance reference of q_1 to p. Since p is a data path, both p and q are marked by Corollary 6.11.8. □

The actual correctness proof of markings is split into two parts. First we will show that for extending reductions the corresponding labelling is a marking for reduct. Thereafter the same will done for projections.

In the sequel, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a $\mathcal{R}$-redex in g. Say $g \overset{\Delta}{\to} h$. Moreover, let use_g be a saturated marking for g, and use_R a marking for R.

The proofs are structured as follows. We distinguish various situations of critical path combinations $(p, a) \wedge (p', a')$ in h, according to the position of a, a' and to the form of p, p' respectively. It will be shown that such critical paths are *well marked*, i.e. in any situation $p * a$ is marked (according to the constructed use_h) if $p * a \lesssim p' * a'$, and $p' * a'$ is marked in case $p' * a' \lesssim p * a$.

In the drawings, dotted lines refer to paths and references in the original graph g.

Extending reductions

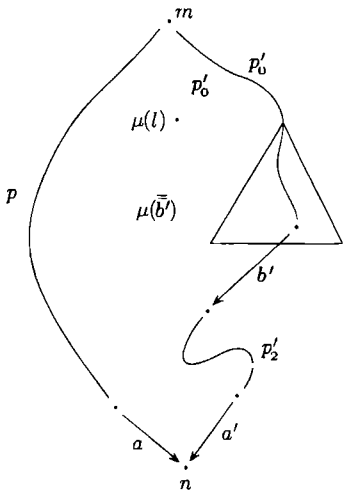
Suppose R is extending. Set $use_h = use_g^{\Delta}$.

6.11 10. PROPOSITION. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a, a' are both old. Then these paths are well marked.*

PROOF. Distinguish cases as to the form of p, p' Say m is the joining node of p, p'.

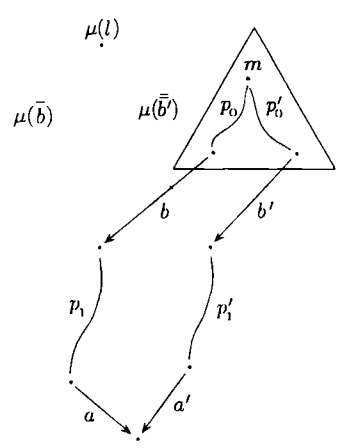
Case 1. p, p' are old. Then we are done.

Case 2. p, p' are new. Suppose $p * a \lesssim p' * a'$; the reverse case is treated similarly. Let b, b' be the border references on p, p' respectively.



Write $p = p_0 * b * p_1$ and $p' = p'_0 * b' * p'_1$. Note that $b \lesssim b'$ in $R \mid r$. Since R is consistent with the argument classification, one has $\bar{b} \lesssim \bar{b'}$, and therefore $\mu(\bar{b}) \lesssim \mu(\bar{\bar{b'}})$ in g. Observe that $\mu(\bar{b}) * p_1 * a \neq \mu(\bar{\bar{b'}}) * p'_1 * a'$, since $a \neq a'$. By Proposition 6.11.5 the path $\mu(\bar{b}) * p_1 * a$ is marked. If $\mu(\bar{b})$ is marked then $use(b) = \otimes$. Otherwise $p_1 * a$. So in both cases $b * p_1 * a$ is marked showing that $p * a$ is marked.

Case 3 Otherwise. Suppose, without loss of generality, p is old and p' is new. Let b' be the border reference on p'.



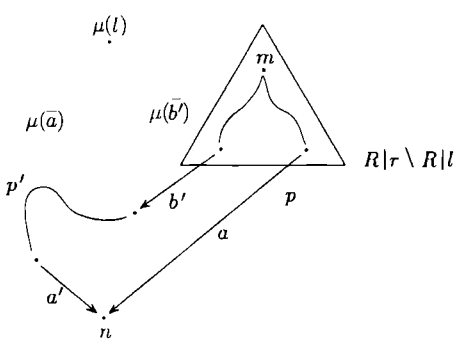
Write $p' = p'_0 * p'_1 * b' * p'_2$ with $p'_0 : m \rightsquigarrow r_R$, $p'_1 : r_R \rightsquigarrow \mathrm{r}(b')$ and $p'_2 : \mathrm{d}(b') \rightsquigarrow a'$. Then in g one has $p'_0 : m \rightsquigarrow \mu(l)$. Consider in g the path $p'_0 * \mu(\bar{\bar{b'}}) * p'_2$. Since $a \neq a'$ one has $p * a \neq p'_0 * \mu(b') * p'_2 * a'$. If $p * a \trianglelefteq p' * a'$ then $p * a$ is marked by Proposition 6.11.5. If $p' * a' \trianglelefteq p * a$ then the same proposition shows that $p'_0 * \mu(\bar{\bar{b'}}) * p'_2 * a'$ is marked in g. Hence $p' * a'$ is marked in h. □

The situation in which the pair a, a' contains a new reference is more involved.

6.11.11. PROPOSITION. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a or a' is new. Then these paths are well marked.*

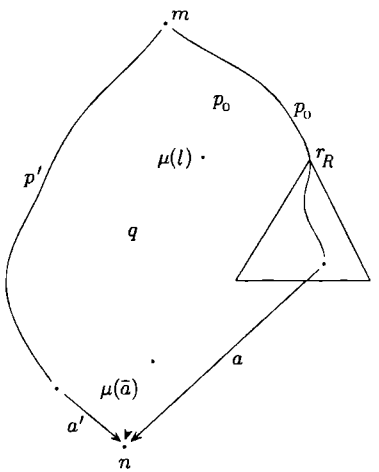
PROOF. Suppose, without loss of generality, that a is new. If a' is also new and $\mathrm{d}_R(a) = \mathrm{d}_R(a')$ then we are done since $R \mid r$ is well marked. If a' is a redirected reference then we only have to consider the case that $\mathrm{d}(a) = r_R$. But then r_R is on a cycle, so $use(R) = \otimes$, so a' is marked in h. Now assume a is a border reference. We proceed by a case distinction. Say m is the joining node of the critical path combination.

Case 1. p' is new. Let b' be the border reference of $p' * a'$.



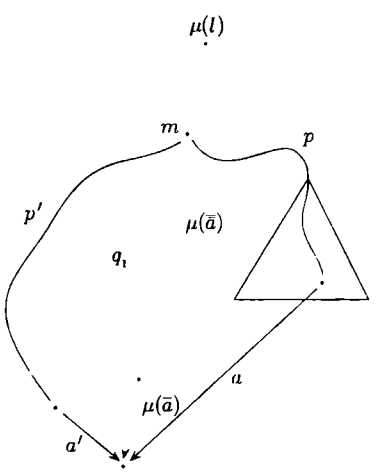
If $p * a \trianglelefteq p' * a'$, then $a \trianglelefteq b'$ in $R \mid r$, and, by consistency of R with the argument classification, $\bar{a} \trianglelefteq \bar{b'}$. Then one has $\mu(\bar{a}) \trianglelefteq \mu(\bar{\bar{b'}})$, and therefore also $\mu(\bar{a}) \trianglelefteq \mu(\bar{\bar{b'}}) * p' * a'$ in g. Now observe that $\mu(\bar{\bar{b'}}) * p' * a', \mu(\bar{a}) : \mu(l) \rightsquigarrow n$. By absence of stacked references one has $\mu(\bar{\bar{b'}}) * p' * a' \neq \mu(\bar{a})$. Then $\mu(\bar{a})$ is marked by Proposition 6.11.5, and therefore $use(a) = \otimes$. If $p' * a' \trianglelefteq p * a$, one similarly concludes that $\mu(\bar{\bar{b'}}) * p' * a'$ is marked. Note that if $\mu(\bar{\bar{b'}})$ is marked then $use(b') = \otimes$. Hence also $b' * p' * a'$ is marked. (Note that case 1 applies if a' is old as well as if a' is new; in the latter case one immediately has $b' \neq a$.)

Case 2. p' is old.



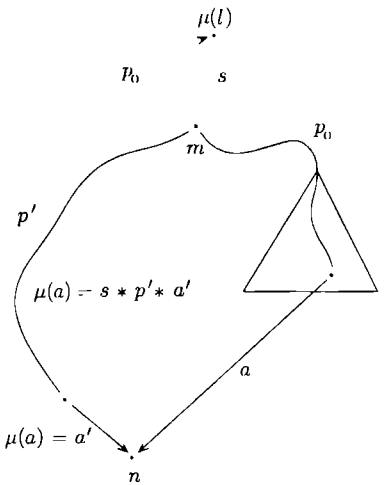
Say $p_0 : m \rightsquigarrow r_R$ is the initial part of p. Then $p_0 : m \rightsquigarrow \mu(l)$ is a path in g. Moreover, $p_0 * \mu(\bar{\bar{a}}) : m \rightsquigarrow n$ is a path in g. If m and n coincide then $p' * a'$ is cyclic. Hence by Proposition 6.11.9 and the fact that $\mu(l) \notin p' * a'$, both $\mu(\bar{\bar{a}})$ and $p' * a'$ are marked in g, so $p * a$ and $p' * a'$ are marked in h. Now assume $m \neq n$.

Case 2a. $p' * a'$ is not a final part of $\mu(\bar{\bar{a}})$. If $m \notin \mu(\bar{\bar{a}})$ set $q_1 = p_0 * \mu(\bar{\bar{a}})$. Otherwise let q_1 be the tail part of $p_0 * \mu(\bar{\bar{a}})$ starting with the last occurrence of m. Note that $q_1 : m \rightsquigarrow n$ and that q_1 is not root cyclic. The case $m \in \mu(\bar{\bar{a}})$ is depicted below.



Observe that that q_1 is not empty (since $m \neq n$), and that $p' * a' \neq q_1$. If $p * a \trianglelefteq p' * a'$ in h then also $q_1 \trianglelefteq p' * a'$ (observe that m is simple in case $m \in \mu(\bar{\bar{a}})$). Hence q_1 is marked by Proposition 6.11.5. Since $\mu(l) \in \mu(\bar{\bar{a}})$ is a function node it follows that $\mu(\bar{\bar{a}})$ is marked, so $use(a) = \otimes$. If $p' * a' \trianglelefteq p * a$ then $p' * a'$ is marked by Proposition 6.11.5.

Case 2b. $\mu(\bar{\bar{a}}) = s * p' * a'$ for some path s.



First note that $\mu(\bar{\bar{a}})$ contains a cycle if $n \in s$. Since $\mu(l)$ is not on this cycle, $\mu(\bar{\bar{a}})$ is marked by Proposition 6.11.9. Assume $n \notin s$.

Claim. In g every path leading to m is marked. Hence by saturation the first reference of $p' * a'$ is marked. Consequently, $p' * a'$ and $\mu(\bar{\bar{a}})$ are marked, and $use(a) = \otimes$, so $p * a$ is marked.

Proof of the claim. Let $t : r_g \rightsquigarrow m$ be a path in g. Suppose s_1, t_1, s_2, t_2, r is the tail factorization of s, t. If s_1 is not empty then t_2 is simple, since $\mu(l) \notin t_2$. By Proposition 6.11.9 t is marked. If s_1 is empty reasoning becomes slightly more delicate. Note that $t_2 * r \parallel s$. We distinguish two cases.

Case 1. In g there exists a path $q : r_g \rightsquigarrow m$ containing a reference $c = (n', i)$ such that $c \notin s$, $n' \neq \mu(l)$ and either $\mathrm{d}(c) \in s$ or $\mathrm{d}(c) = m$. Observe that q, s can be written as $q = q' * c * r'$ and $s = s' * c' * r'$ respectively, with $c \neq c'$. By Proposition 6.11.9 the path $p_0 * t_2 * r$ is marked (consider either $c * r'$ or $c' * r'$). Hence t is marked since t_2 starts with the function node $\mu(l)$.

Case 2. Otherwise. Say $[\bar{\bar{a}}] = j$; write $s = (\mu(l), j) * s'$. Now observe that t can be written as $t = t' * (\mu(l), i) * s'$ for some i. Since m is present in h, there is a border reference c in h such that either $\mathrm{d}(c) \in s$ or $\mathrm{d}(c) = m$. Note that $c \neq a$. Say $[\bar{c}] = k$. Then $\mu(\bar{c}) = (\mu(l), k) * s''$ for some initial part s'' of s'. By absence of stacked references one has $j \neq k$. Moreover a and c are stack suspect. Hence

by Lemma 6.10.7 (applied twice) one has $\imath \lesssim \jmath$ and $\imath \lesssim k$. Since either $\imath \neq \jmath$ or $\imath \neq k$ the first reference $(\mu(l), \imath)$ of $t_2 * r$ is marked, so t is marked. □

This completes the proof of correctness of $use_h = use_g^{\Delta}$

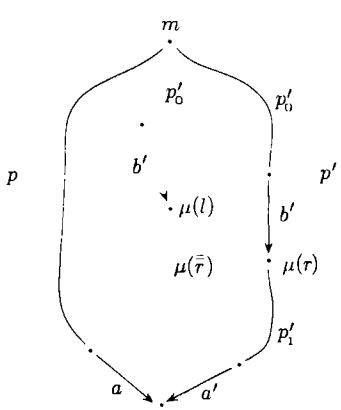
Projecting reductions

Suppose R is projecting. Set $use_h = use_g^{\Delta}$. Although projections look simpler than extending reductions, the analysis requires a more complex case distinction.

First of all, if $\mu(l) = \mu(r)$, then the redirection is trivial and we are done immediately. Assume, for the rest of this proof, that $\mu(l) \neq \mu(r)$. Since moreover the pattern of R does not contain function nodes apart from l, the path $\mu(\bar{r})$ is not root cyclic.

6.11.12. PROPOSITION. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a, a' are both old. Then these paths are well marked.*

PROOF. Say m is the joining node of the critical path combination. If both paths are old and $p * a \lesssim p' * a'$ then trivially $p * a$ is marked. Now suppose p is old, and p' is new. Write $p' = p'_0 * b' * p'_1$ with $p'_1 : \mu(r) \rightsquigarrow \mathrm{r}(a')$.



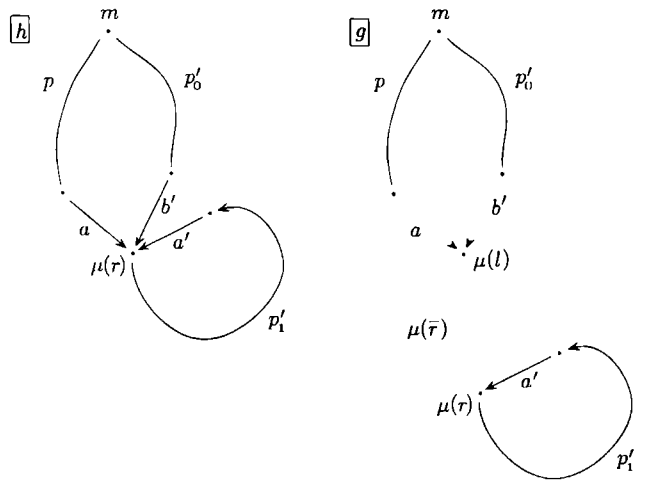
Case 1. $p * a \lesssim p' * a'$ in h. Then in g one has $p * a \lesssim p'_0 * b' * \mu(\bar{r}) * p'_1 * a'$. Hence $p * a$ is marked by Proposition 6.11.5.

Case 2. $p' * a' \lesssim p * a$ in h. Then $p'_0 * b' * \mu(r) * p'_1 * a'$ is marked, again by Proposition 6.11.5. If $p'_1 * a'$ is marked we are done. Otherwise $\mu(\bar{r})$ is marked in g since it starts with a function node and therefore b' is marked in h, showing that $p' * a'$ is marked. □

6.11.13. PROPOSITION. *Let $(p, a) \wedge (p', a')$ be a critical path combination, where $a \neq a'$, $\mathrm{d}_h(a) = \mathrm{d}_h(a')$, and a or a' is new. Then these paths are well marked.*

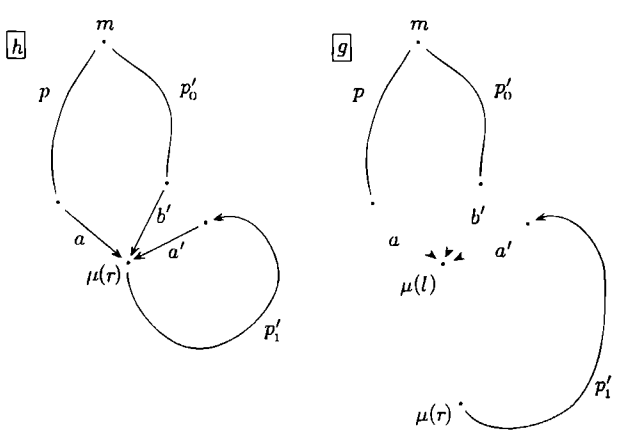
PROOF. Say m is the joining node of $(p, a) \wedge (p', a')$.

Case 1. a is new, a' is old, p is old and p' is new. Write $p' = p'_0 * b' * p'_1$ where b' is the new reference on p'. Consider in g the paths $\mu(\bar{r})$ and $\mu(\bar{\bar{r}}) * p'_1 * a'$. This looks as follows.



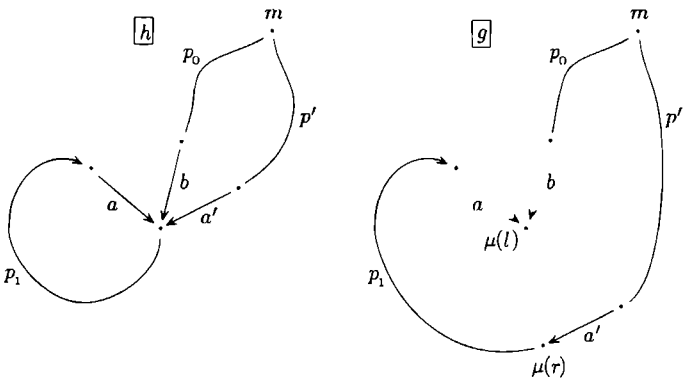
Since $\mu(l) \notin p'_1$ the path $\mu(\bar{\bar{r}})$ contains a reference not appearing on the cycle $p'_1 * a'$. By Proposition 6.11.9 both of these paths are marked. Since $\mu(\bar{\bar{r}})$ is marked, also $p * a$ is marked in h. Moreover because $\mu(\bar{r}) * p'_1 * a'$ is marked it follows that $(p'_0 *)\ b' * p'_1 * a'$ is marked in h.

Case 2. a, a' are both new, and one of p, p' is new. Assume, without loss of generality, that p is old and p' is new. Again write $p' = p'_0 * b' * p'_1$. The situation in h is the same as in case 1. In g, however, one has the following.



By Proposition 6.11.7 both $p * a$ and $\mu(\overline{r}) * p_1' * a'$ are marked in g, since a and a' start in different nodes. Hence $p * a$ and $p' * a'$ are marked in h.

Case 3. a is new, a' is old, p is new and p' is old. Write $p = p_0 * b * p_1$ with b new. The situation is as follows.

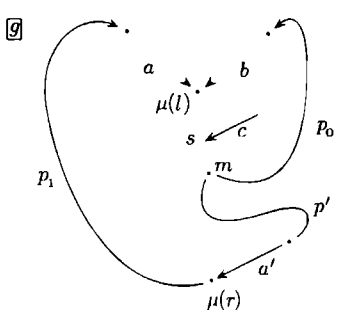


As to $p * a$, observe that a and b start in distinct nodes since p is acyclic. Therefore a is marked by Proposition 6.11.7.

As to $p' * a'$, if $m = \mu(r)$ then $p' * a'$ is marked by Proposition 6.11.9 (consider $\mu(\overline{r})$ and $p' * a'$). Now suppose $m \neq \mu(r)$.

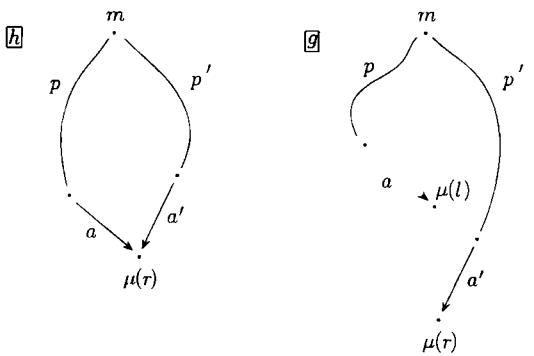
Case a. $p' * a'$ is not a tail part of $\mu(\overline{r})$. Then we are done by Proposition 6.11.9 (consider $p' * a'$ and $p_1 * a * \mu(\overline{r})$).

Case b. $p' * a'$ is a tail part of $\mu(\overline{r})$. Write $\mu(\overline{r}) = s * p' * a'$.

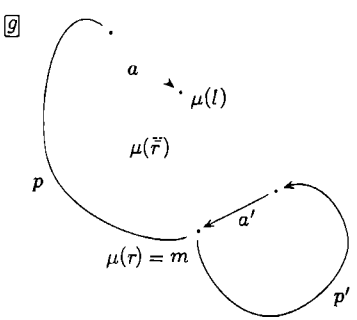


Since m is present in h there exists a reference c to s, starting in a node not occurring on s. Note that c is old since $m \neq \mu(r)$. This shows that s is marked (consider the cycle $s * p' * a' * p_1 * a$ and use Proposition 6.11.7). From this we can deduce, as in the proof of Proposition 6.11.11 (case 2b) but easier, that any rooted path to data node m is marked. Hence by saturation the first reference of $p' * a'$ is marked and we are done.

Case 4. a is new, a' is old, and p, p' are old.



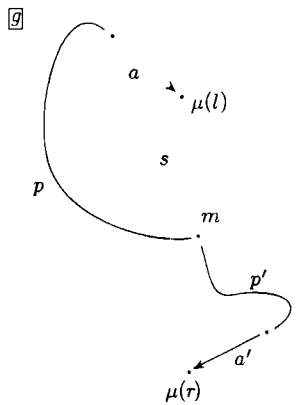
If $m = \mu(r)$ then the following situation appears in g.



Now both $\mu(\bar{r})$ and $p' * a'$ are marked by Proposition 6.11.9. Hence in h the paths $p * a$ and $p' * a'$ are marked. Now suppose $m \neq \mu(r)$.

Case a. $p' * a'$ is not a tail part of $\mu(\bar{r})$. Let q be the unique subpath of $p * a * \mu(\bar{r})$ such that $q : m \rightsquigarrow \mu(r)$ and q is not root cyclic. If $p * a \trianglelefteq p' * a'$ in h then $q \trianglelefteq p' * a'$ in g (observe that m is simple if it occurs on $\mu(\bar{r})$). Hence by Proposition 6.11.5 (note that q and $p' * a'$ only intersect in data nodes) q is marked in g, so $p * a$ is marked in h. In case $p' * a' \trianglelefteq p * a$ a similar argument shows that $p' * a'$ is marked.

Case b. $p' * a'$ is a tail part of $\mu(\overline{r})$.



First note that m is reachable from r_h, so there exists a reference c to a node of s (not equal to $\mu(l)$) starting in a node not occurring on s. Hence s is marked by Proposition 6.11.7, so a is marked in h, so $p * a$ is marked. As to $p' * a'$, as before one can show that the first reference of $p' * a'$ is marked by saturation.

Case 5. a and a' are new, and p, p' are old. Then also in g one has $p * a \lesssim p' * a'$ or $p' * a' \lesssim p * a$ respectively, so $p * a$ (or $p' * a'$ respectively) is marked. □

Thus it is shown that use_g^{Δ} is a marking for h.

6.12. Typing Reducts

In this section we will complete the proof that uniqueness typing is preserved during reduction.

In the proofs of this section, let $\mathfrak{T}$ be Curry safe for $\mathcal{F}$. Let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a redex in g. Say $g \xrightarrow{\Delta} h$. Let $\mathcal{U}_g$ be a $\mathcal{F}, \mathcal{A}$-uniqueness typing for g according to use_g.

The key result is the following.

6.12.1. PROPOSITION. *Suppose use_g is saturated. Then there exists a marking use_h for h and a $\mathcal{F}, \mathcal{A}$-uniqueness typing $\mathcal{U}_h$ according to use_h such that $\mathcal{U}_g(r_g) = \mathcal{U}_h(r_h)$*

The proof will be split into two parts. We start with some results on coercions.

6.12.2. LEMMA. *Let $u, v \in \mathbb{M}$. Then*

(i) $\sigma \leq^u \tau,\ \tau \leq^v \rho \Rightarrow \sigma \leq^{u+v} \rho$.

(ii) $\sigma \leq \tau,\ \tau \leq^v \rho \Rightarrow \sigma \leq^v \rho$.

PROOF. Straightforward. □

6.12.3. LEMMA. $\sigma \leq \tau$, $[\sigma] = \times \Rightarrow \sigma \leq^{\otimes} \tau$.

PROOF. Simple. □

Typings for extending reductions

6.12.4. DEFINITION. Using Corollary 6.7.6, determine $\mathcal{U}$ and *use* for $R \mid r$ such that

$$\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l)) \tag{1}$$

and for any $n \in (R \mid l) \cap (R \mid r)$

$$\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n), \tag{2}$$

$$[\mathcal{U}(n)] \neq \times \Rightarrow \mu(\overline{n}) \text{ is not marked by } use_g. \tag{3}$$

(i) Set $use_h = use^{\Delta}$ (see Definition 6.10.4).
(ii) Define $\mathcal{U}_h = \mathcal{U}^{\Delta}$ as follows.

$$\begin{aligned} \mathcal{U}^{\Delta}(n) &= \mathcal{U}_g(n) \quad \text{if } n \in g, \\ &= \mathcal{U}(n) \quad \text{if } n \in (R \mid r) \backslash (R \mid l). \end{aligned}$$

Note that use_h is a marking for h, by Theorem 6.10.8.

6.12.5. LEMMA. *Let* $\sigma \in \widehat{\mathbb{T}}$, *and* $u \in \mathbb{M}$.
(i) $\mathcal{U}_g(\mu(l)) \leq^u \sigma \Rightarrow \mathcal{U}(r) \leq^{u+use(R)} \sigma$.
(ii) *For each border reference* a *of* R

$$\mathcal{U}(\mathrm{d}(a)) \leq^u \sigma \Rightarrow \mathcal{U}_g(\mu(\mathrm{d}(a)) \leq^{u+use(\mu(\bar{\bar{a}}))} \sigma.$$

PROOF. (i) Note that

$$\begin{aligned} \mathcal{U}(r) &\leq^{use(R)} \mathcal{U}_g(\mu(l)), \quad \text{by (1)} \\ &\leq^u \sigma, \quad \text{by assumption.} \end{aligned}$$

Hence by Lemma 6.12.2 (i) we are done.
(ii) Observe that

$$\begin{aligned} \mathcal{U}_g(\mu(\mathrm{d}(a)) &\leq \mathcal{U}(\mathrm{d}(a)), \quad \text{by (2)} \\ &\leq^u \sigma, \quad \text{by assumption.} \end{aligned}$$

In case $[\mathcal{U}(\mathrm{d}(a))] = \times$ the result follows from the lemmas 6.12.3 and 6.12.2 (ii). If, on the other hand, $[\mathcal{U}(\mathrm{d}(a))] \neq \times$ then $use_g(\mu(\bar{\bar{a}})) = \odot$ by (3), so we are done by Lemma 6.12.2 (ii). □

6.12.6. LEMMA. *$\mathcal{U}_h$ is a uniqueness typing according to use_h.*

PROOF. Since the node types are taken from $\mathcal{U}_g$ and $\mathcal{U}$ we only have to check that the coercions induced by the new use_h are valid. If $use_h(a)$ is equal to $use_g(a)$ or $use(a)$ this is simple. In other cases (redirected and border references) the correctness of $\mathcal{U}_h$ follows from Lemma 6.12.5. □

Finally note that $\mathcal{U}_h(r_h) = \mathcal{U}_g(r_g)$ since $r_g = r_h$: the root does not take part in the rewriting process. This completes the proof of Proposition 6.12.1 in the extension case.

Typings for projections

6.12.7. DEFINITION. Using Corollary 6.7.6, determine $\mathcal{U}$ and use for $R \mid r$ such that

$$\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l)) \tag{1}$$

and for any $n \in (R \mid l) \cap (R \mid r)$

$$\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n), \tag{2}$$

$$[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(\bar{\bar{n}}) \text{ is not marked by } use_g. \tag{3}$$

(i) Set $use_h = use^{\Delta}$ (see Definition 6.10.10).
(ii) Define $\mathcal{U}_h = \mathcal{U}^{\Delta} = \mathcal{U}_g \restriction N_h$.

First note that use_h is a marking function for h, by Theorem 6.10.11.

6.12.8. LEMMA. *Let $\sigma \in \hat{\mathbb{T}}$, and $u \in \mathbb{M}$. Then*

$$\mathcal{U}_g(\mu(l)) \leq^u \sigma \;\Rightarrow\; \mathcal{U}_g(\mu(r)) \leq^{u + use(\mu(\bar{\bar{r}}))} \sigma.$$

PROOF. Analogous to the proof of Lemma 6.12.5 (ii). □

6.12.9. LEMMA. *$\mathcal{U}_h$ is a uniqueness typing for h according to use_h.*

PROOF. By the fact that $\mathcal{U}_g$ is a uniqueness typing for g, and Lemma 6.12.8 (showing that the modifications made to use_g are harmless). □

The subject reduction property

Finally we can state and prove the main result.

6 12.10. SUBJECT REDUCTION THEOREM. *Suppose $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ is Curry safe for $\mathcal{F}$ Then for any $g \in \mathcal{G}$*

$$\mathcal{F}, \mathcal{A} \vdash g : \sigma, \quad g \xrightarrow{}_{\mathcal{R}} h \;\Rightarrow\; \mathcal{F}, \mathcal{A} \vdash h : \sigma.$$

PROOF. Say $\mathcal{U}_g$ is a $\mathcal{F}, \mathcal{A}$-uniqueness typing for g according to use_g. By the Saturation Theorem 6.9.10 there exists a saturated marking function use'_g and a typing $\mathcal{U}'_g$ according to use'_g such that $\mathcal{U}'_g(r_g) = \mathcal{U}_g(r_g)$. Now Proposition 6.12.1 applies and we are done. □

6.13. Applications

In this section we will give some examples that show how the two problems mentioned in the introduction can be solved by using uniqueness types.

Consider the following list reversing function which can be implemented as a 'destructive' function if the given uniqueness type is used.

$\mathbf{Rev} : \mathrm{List}^{\bullet}(\alpha^{x}) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{x})$	$\mathbf{Rev}\ (l) \rightarrow \mathbf{H}(l, \mathbf{Nil})$
$\mathbf{H} : (\mathrm{List}^{\bullet}(\alpha^{x}), \mathrm{List}^{\bullet}(\alpha^{x})) \rightarrowtail \mathrm{List}^{\bullet}(\alpha^{x})$	$\mathbf{H}\ (\mathbf{Nil}, r) \rightarrow r$ $\mathbf{H}\ (\mathbf{Cons}(h, t), r) \rightarrow \mathbf{H}(t, \mathbf{Cons}(h, r))$

Note that not only the space of the obsolete **Cons** can be re-used, but also parts of its contents. In fact it even suffices to redirect the reference to t such that it points to r.

The second example shows how a predefined function **WriteChar** can be typed, having as input an object of type File and a character that should be written to the given file. The output consists of the modified file which is also unique, so it can be used for further writing.

$$\mathbf{WriteChar} : (\mathrm{File}^{\bullet}, \mathrm{Char}^{x}) \rightarrowtail \mathrm{File}^{\bullet}$$

A more elaborated example, presenting an efficient quicksort algorithm that performs the sorting *in situ* on the data structure, can be found in Smetsers et al. (1994). This algorithm is based on the same idea as used in the list reversal example.

6.14. Future Research

The present uniqueness type system is still subject to research. It is expected that the constraints with respect to reference stacking (see Section 6.10) can be relaxed using a more refined marking procedure.

The uniqueness type system can be extended to deal with uniqueness attribute *variables*. This allows the types to be generic: a new uniqueness type consists of a type scheme and a collection of 'coercion constraints' on the occurring attribute variables. Theoretical research is performed on type derivation in the extended uniqueness type system.

Furthermore, the relation between reduction strategies and the symbol classification can be concretized further.

Part III

Numerations

Chapter 7

Notes on Fixed Points

7.1. Introduction

In this chapter we use J L Eršov's numeration theory to study some fixed point results in untyped lambda calculus and in recursion theory

The often mentioned analogy between the proofs of the (second) recursion theorem and the fixed point theorem in lambda calculus can be refined In fact, in the context of numeration theory, the fixed point theorem in lambda calculus corresponds to the first recursion theorem, and the (code dependent) second fixed point theorem in lambda calculus corresponds to the second recursion theorem

This chapter contains an introduction to numeration theory, and a survey of results in lambda calculus and recursion theory, all formulated in the enumeration framework The last section shows that the terminology of numerations can be used to give an elegant exposition of the effective version of the first recursion theorem

7.2. Examples

Let us have a look at two familiar fixed point results

Firstly, the fixed point theorem in lambda calculus and its traditional proof run as follows

7 2 1 THEOREM *Let* $F \in \Lambda$ *Then*

$$\exists X \in \Lambda \ FX =_\beta X$$

PROOF Let $F \in \Lambda$ Set $W \equiv \lambda x\, F(xx)$ and $X \equiv WW$ Then

$$\begin{aligned} X &\equiv WW \\ &=_\beta F(WW) \\ &\equiv FX \ \square \end{aligned}$$

A second example comes from arithmetic. Below, we use the traditional denotations for encodings ($\sharp$) and numerals ($\ulcorner\ \urcorner$). We use $\ulcorner\varphi\urcorner$ as an abbreviation for $\ulcorner\sharp\varphi\urcorner$.

7.2.2. THEOREM (Gödel's fixed point theorem).

$$\forall\psi(x)\ \exists\varphi\ \mathbf{PA} \vdash \psi(\ulcorner\varphi\urcorner) \leftrightarrow \varphi.$$

PROOF. There exists a (recursive) function Diag : $\mathbb{N} \to \mathbb{N}$ such that for all $\psi = \psi(x)$

$$\mathrm{Diag}(\sharp\psi) = \sharp\psi(\ulcorner\psi\urcorner).$$

Diag is representable in **PA**, i.e. for some formula $D \equiv D(x, y)$ one has for all $n, k \in \mathbb{N}$

$$\mathrm{Diag}(n) = k \ \Rightarrow\ \mathbf{PA} \vdash \forall y\ [D(\ulcorner n\urcorner, y) \leftrightarrow y = \ulcorner k\urcorner].$$

Now define

$$\begin{aligned} \sigma(x) &\equiv \exists y\ [D(x, y)\ \&\ \psi(y)], \\ \varphi &\equiv \sigma(\ulcorner\sigma\urcorner). \end{aligned}$$

Then $\mathrm{Diag}(\sharp\sigma) = \sharp\varphi$, so

$$\mathbf{PA} \vdash \forall y\ [D(\ulcorner\sigma\urcorner, y) \leftrightarrow y = \ulcorner\varphi\urcorner].$$

Hence in **PA** one has

$$\begin{aligned} \varphi &\leftrightarrow \exists y\ [D(\ulcorner\sigma\urcorner, y)\ \&\ \psi(y)] \\ &\leftrightarrow D(\ulcorner\sigma\urcorner, \ulcorner\varphi\urcorner)\ \&\ \psi(\ulcorner\varphi\urcorner) \\ &\leftrightarrow \psi(\ulcorner\varphi\urcorner). \ \square \end{aligned}$$

Note that in Gödel's fixed point theorem the codes of formulas play an essential role, whereas the lambda calculus theorem is 'code free'. There are, however, similarities with respect to the use of diagonalization and composition properties. Smullyan (1985) formulates these properties for arbitrary applicative structures, in the following way

7.2.3. PROPOSITION. *Let* $\langle X, \cdot\rangle$ *be an applicative structure such that*

$$\exists d \in X\ \ \forall x \in X\ \ d \cdot x = x \cdot x,$$

$$\forall f \in X\ \ \forall g \in X\ \ \exists h \in X\ \ \forall x \in X\ \ h \cdot x = f \cdot (g \cdot x).$$

Then

$$\forall f \in X\ \ \exists x \in X\ \ f \cdot x = x.$$

PROOF. Determine a diagonalizing element d, and $w \in X$ such that

$$\forall x \in X \;\; wx = f \cdot (d \cdot x).$$

Define $x = w \cdot w$. Then

$$\begin{aligned} x &= w \cdot w \\ &= f \cdot (d \cdot w) \\ &= f \cdot x. \;\square \end{aligned}$$

In the next section we describe an alternative approach that applies, however, only to subsystems of lambda calculus and arithmetic.

7.3. Numerations

The theory of numerations has been introduced by Eršov (1973). Below, **PR** stands for the class of partial recursive functions, and **R** for its subclass of total recursive functions. Below, φ ranges over partial and f over total functions.

7.3.1. DEFINITION. (i) A *numeration* is a pair

$$\gamma = (\mathcal{X}, \nu)$$

where $\mathcal{X}$ is a set and $\nu : \mathbb{N} \twoheadrightarrow \mathcal{X}$ a surjection.

(ii) If $\nu(n) = x$ then n is said to be a *code* for x.

(iii) A numeration γ induces an equivalence relation $\sim_\gamma$ on $\mathbb{N}$, defined by

$$m \sim_\gamma n \;\Leftrightarrow\; \nu(m) = \nu(n)$$

In these notes we consider special numerations: the so called precomplete ones.

7.3.2. DEFINITION. Let γ be a numeration. Then γ is *precomplete* if

$$\forall \varphi \in \mathbf{PR} \;\; \exists f \in \mathbf{R} \;\; \forall n \in \mathrm{dom}(\varphi) \;\; f(n) \sim_\gamma \varphi(n).$$

If $\forall n \in \mathrm{dom}(\varphi) \;\; f(n) \sim_\gamma \varphi(n)$, then we say (following Visser (1980)) that f *totalizes* φ *modulo* γ.

If γ is precomplete, this totalization can be done uniformly.

7.3.3. PROPOSITION. *Let γ be a precomplete numeration. Then there exists a recursive function $c_\gamma : \mathbb{N}^2 \to \mathbb{N}$ such that for each $e \in \mathbb{N}$*

$$\lambda n.c_\gamma(e, n) \quad \textit{totalizes} \quad \{e\} \quad \textit{modulo } \gamma.$$

PROOF. Consider the 'universal function' U on encoded pairs of natural numbers:

$$U(\langle e, x\rangle) \simeq \{e\}(x).$$

Let C totalize U modulo γ; define

$$c_\gamma(e, n) = C(\langle e, n\rangle). \ \square$$

Note that the above construction depends on C. In the applications below, we will assume some particular C to be chosen and denote $\boldsymbol{\lambda} n.c_\gamma(e, n)$ by $\{e\}^\gamma$.

A first example of a precomplete numeration is given by the closed term model of (untyped) λ-calculus.

7.3.4. DEFINITION. (i) Let $\mathbf{E}$ be a λ-calculus enumerator (cf. Barendregt (1991)), such that

$$\mathbf{E}\ulcorner M\urcorner =_\beta M$$

for all closed M. As usual, $\ulcorner M\urcorner$ stands for $\ulcorner \sharp M\urcorner$.

(ii) The numeration $\mathcal{L}_\beta$ is the structure $\langle \mathbf{L}_\beta,\ e\rangle$, where

$$\begin{aligned} \mathbf{L}_\beta &= \Lambda^\circ \Big/ =_\beta \ = \{[M] \mid M \in \Lambda^\circ\}, \\ e(n) &= [\mathbf{E}\ulcorner n\urcorner]. \end{aligned}$$

7.3.5. PROPOSITION (H.P. Barendregt). *The numeration $\mathcal{L}_\beta$ is precomplete.*

PROOF. Let $\varphi \in \mathbf{PR}$. Determine a term $F \in \Lambda^\circ$ that λ-defines φ with respect to the numerals $\ulcorner n\urcorner$. Take

$$f(n) = \sharp\mathbf{E}(F\ulcorner n\urcorner).$$

Then $f \in \mathbf{R}$ by effectiveness of $\ulcorner\ \urcorner$ and application. Now suppose $\varphi(n)\downarrow$. Then

$$\begin{aligned} e(f(n)) &= [\mathbf{E}\ulcorner f(n)\urcorner] \\ &= [\mathbf{E}\ulcorner \mathbf{E}(F\ulcorner n\urcorner)\urcorner] \\ &= [\mathbf{E}(F\ulcorner n\urcorner)] \\ &= [\mathbf{E}(\ulcorner \varphi(n)\urcorner)], \quad \text{since } F \ \lambda\text{-defines } \varphi \\ &= e(\varphi(n)). \ \square \end{aligned}$$

7.3.6. DEFINITION. Let $\gamma = \langle \mathcal{X}, \nu\rangle$ be a numeration. Then γ is *complete* if for some $a \in \mathcal{X}$

$$\forall \varphi \in \mathbf{PR}\ \exists f \in \mathbf{R}\ \forall n \in \mathbb{N}\ [\ (n \in \mathrm{dom}(\varphi) \ \Rightarrow\ f(n) \sim_\gamma \varphi(n)\ \&\ (n \notin \mathrm{dom}(\varphi) \ \Rightarrow\ \nu(f(n)) = a\ ].$$

We will call such an a a *special element* of γ.

7.3.7. PROPOSITION. *The numeration $\mathcal{L}_\beta$ is not complete.*

PROOF. Remember that $=_\beta$ is re, so any $\sim_{\mathcal{L}_\beta}$-class is re. Suppose, towards a contradiction, that $[A]$ is a special element, say $e(a) = [A]$. Choose $b \not\sim_{\mathcal{L}_\beta} a$. Let K be an re set that is not recursive. Define the function

$$\begin{aligned} \varphi(x) &\simeq b \quad \text{if } x \in K, \\ &\simeq \uparrow \quad \text{otherwise.} \end{aligned}$$

Suppose f totalizes φ Let $n \in \mathbb{N}$. Then

$$x \notin K \Leftrightarrow f(n) \sim_{\mathcal{L}_\beta} a.$$

Note that the right-hand side is re, so K is co-re, thus contradicting the assumption that K is not recursive. □

We now introduce a complete numeration: that of partial recursive functions.

7.3.8. DEFINITION. The *numeration of partial recursive functions* (notation $\mathcal{PR}$) is defined by

$$\mathcal{PR} = \langle \mathbf{PR}, \{\ \} \rangle.$$

7.3.9. PROPOSITION. *The numeration $\mathcal{PR}$ is complete.*

PROOF. Let $\varphi \in \mathbf{PR}$. Define $\psi : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ by

$$\psi(e, x) \simeq \{\varphi(e)\}(x) \quad (\simeq \uparrow \text{ if } \varphi(e)\uparrow).$$

Let t be an index of ψ. Define $f(e) = S_1^1(t, e)$. Then $f \in \mathbf{R}$ and

$$\begin{aligned} \psi(e, x) &\simeq \{t\}(e, x) \\ &\simeq \{f(e)\}(x). \end{aligned}$$

Then f totalizes φ modulo $\mathcal{PR}$. The everywhere undefined function $\boldsymbol{\lambda} n.\uparrow$ is a special element. □

The following abstract fixed point theorem is due to J.L. Eršov.

7.3.10. THEOREM. *Let γ be precomplete. Then*

$$\forall \varphi \in \mathbf{PR}\ \exists n \in \mathbb{N}\ [\varphi(n)\downarrow \Rightarrow \varphi(n) \sim_\gamma n].$$

PROOF. Using Proposition 7.3.3, determine an index w for the function

$$\boldsymbol{\lambda} x.\varphi(\{x\}^{\gamma}(x)).$$

Define $n = \{w\}^{\gamma}(w)$. Suppose $\varphi(n)\downarrow$. Then

$$\begin{array}{rcl} \varphi(n) & = & \varphi(\{w\}^{\gamma}(w)) \\ & \simeq & \{w\}(w) \\ & \sim_{\gamma} & \{w\}^{\gamma}(w) \\ & \simeq & n. \ \square \end{array}$$

We will usually apply this to total functions.

7.3.11. COROLLARY. *Let γ be precomplete. Then*

$$\forall f \in \mathbf{R} \ \exists n \in \mathbb{N} \ f(n) \sim_{\gamma} n.$$

Also for the fixed point theorem there is an effective version.

7.3.12. PROPOSITION. *Let γ be precomplete. Then there is a function $g \in \mathbf{R}$ such that for each $e \in \mathbb{N}$*

$$\{e\}(g(e))\downarrow \ \Rightarrow \ \{e\}(g(e)) \sim_{\gamma} g(e).$$

PROOF. By observing that the index w in the above proof can be found from an index of φ, and n can be found effectively from w. $\square$

7.4. Code Dependent Fixed Point Results

We can apply Eršov's fixed point theorems to obtain versions of the 'second' fixed point theorems.

We start with a result in untyped lambda calculus.

7.4.1. PROPOSITION.

$$\forall f \in \mathbf{R} \ \exists n \in \mathbb{N} \ \mathbf{E}^{\ulcorner} f(n)^{\urcorner} =_{\beta} \mathbf{E}^{\ulcorner} n^{\urcorner}.$$

PROOF. By Corollary 7.3.11 and Proposition 7.3.5. $\square$

This can easily be reformulated into a more familiar statement, cf. Barendregt (1984).

7.4.2. COROLLARY.

$$\forall F \in \Lambda^{\circ} \ \exists X \in \Lambda^{\circ} \ F^{\ulcorner} X^{\urcorner} =_{\beta} X.$$

PROOF. Let $F \in \Lambda^\circ$. Define $f \in \mathbf{R}$ by

$$f(n) = \sharp F^\ulcorner \mathbf{E}^\ulcorner n^\urcorner\urcorner.$$

Determine $n_0 \in \mathbb{N}$ such that $\mathbf{E}^\ulcorner f(n_0)^\urcorner =_\beta \mathbf{E}^\ulcorner n_0^\urcorner$. Take $X \equiv \mathbf{E}^\ulcorner n_0^\urcorner$. Then

$$\begin{aligned} X &\equiv \mathbf{E}^\ulcorner n_0^\urcorner \\ &=_\beta \mathbf{E}^\ulcorner f(n_0)^\urcorner \\ &=_\beta \mathbf{E}^\ulcorner F^\ulcorner \mathbf{E}^\ulcorner n_0^\urcorner\urcorner\urcorner \\ &=_\beta F^\ulcorner \mathbf{E}^\ulcorner n_0^\urcorner\urcorner \\ &\equiv F^\ulcorner X^\urcorner. \ \square \end{aligned}$$

For recursion theory we obtain the following.

7.4.3. PROPOSITION. $\forall f \in \mathbf{R}\ \exists n \in \mathbb{N}\ \{f(n)\} = \{n\}$.

PROOF. By Corollary 7.3.11 and Proposition 7.3.9. $\square$

From this the second recursion theorem in S.C. Kleene's formulation follows easily.

7.4.4. COROLLARY *Let $\psi : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ be partial recursive. Then*

$$\exists e \in \mathbb{N}\ \forall n \in \mathbb{N}\ \{e\}(n) \simeq \psi(e, n).$$

PROOF. Write $\psi(e, n) \simeq \{f(e)\}(n)$ using the S-m-n theorem. $\square$

It is also possible to derive a version of Gödel's fixed point theorem, in a restricted version (like in λ-calculus). In the λ-calculus a 'self-interpreter' only exists for classes of terms over a fixed set of free variables. In arithmetic this corresponds to the existence of a *truth predicate* for sets of formulas with a restricted quantifier depth. Below the numerations corresponding to systems of arithmetic are described; the construction is taken from Visser (1980).

7.4.5. DEFINITION. Let $n \in \mathbb{N}$, $n \geq 1$.

(i) Let $\mathbf{PA}(\Sigma^0_n(\vec{x}))$ be the set of formulas of Peano arithmetic of Σ^0_n-form, with free variables in $\vec{x}$.

(ii) Let $\psi, \chi \in \mathbf{PA}(\Sigma^0_n(\vec{x}))$. Then ψ and χ are $\mathbf{PA}$-*equivalent* (notation $\psi \Leftrightarrow_{\mathbf{PA}} \chi$) if

$$\mathbf{PA} \vdash \forall \vec{x}\ [\psi(\vec{x}) \leftrightarrow \chi(\vec{x})].$$

(iii) Let $(\psi_k)_{k \in \mathbb{N}}$ be some (standard) enumeration of formulas of $\mathbf{PA}(\Sigma^0_n(\vec{x}))$. The numeration of $\Sigma^0_n(\vec{x})$-arithmetic is the system

$$\mathcal{A}^{\vec{x}}_n = (\mathbf{PA}(\Sigma^0_n(\vec{x}))/\Leftrightarrow_{\mathbf{PA}},\ \boldsymbol{\lambda} k.[\psi_k]).$$

7.4.6. FACT. For each $n \geq 1$ and each $\vec{x}$ there is a predicate $\mathrm{Tr}_n(y, \vec{x})$ such that for all formulas $\psi \equiv \psi(\vec{x})$ of $\mathbf{PA}(\Sigma^0_n(\vec{x}))$

$$\mathbf{PA} \vdash \forall \vec{x}\, [\psi(\vec{x}) \leftrightarrow \mathrm{Tr}_n(\ulcorner \psi(\vec{x}) \urcorner, \vec{x})].$$

7.4.7. PROPOSITION. *$\mathcal{A}^{\vec{e}}_n$ is precomplete.*

PROOF. Let $\varphi \in \mathbf{PR}$. Using the representation of the graph of φ in $\mathbf{PA}$, define

$$f(m) = \sharp(\exists y\, [\text{“}\varphi(m) \simeq y\text{”} \,\&\, \mathrm{Tr}_n(y, \vec{x})]).$$

Then $f \in \mathbf{R}$. Now suppose $\varphi(n)\downarrow$. Then in $\mathbf{PA}$ one has

$$\begin{aligned} \psi_{f(m)} &\equiv \exists y\, [\text{“}\varphi(m) \simeq y\text{”} \,\&\, \mathrm{Tr}_n(y, \vec{x})] \\ &\leftrightarrow \mathrm{Tr}_n(\ulcorner \psi_{\varphi(m)} \urcorner, \vec{x}) \\ &\leftrightarrow \psi_{\varphi(m)}. \ \square \end{aligned}$$

7.5. Fixed Points of Endomorphisms

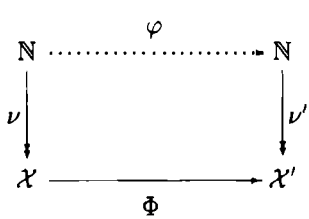
Some fixedpoint theorems which are in a sense 'code free' can be proved using numeration theory. We first introduce some auxiliary notions.

7.5.1. DEFINITION. Let $\gamma = \langle \mathcal{X}, \nu \rangle$ and $\gamma' = \langle \mathcal{X}', \nu' \rangle$ be numerations.

(i) Let $\Phi : \mathcal{X} \rightharpoonup \mathcal{X}'$. Φ is a *partial morphism* from γ to γ' (notation $\Phi : \gamma \rightharpoonup \gamma'$) if

$$\exists \varphi \in \mathbf{PR}\ \forall n \in \mathbb{N}\ [\Phi(\nu(n))\downarrow \ \Rightarrow\ \nu'(\varphi(n)) = \Phi(\nu(n))],$$

in a diagram:



In this case φ is said to be a *lifting* of Φ.

(ii) Let $\Phi : \mathcal{X} \to \mathcal{X}'$. Φ is a *morphism* from γ to γ' (notation $\Phi : \gamma \to \gamma'$) if $\Phi : \gamma \rightharpoonup \gamma'$ and Φ is total, thus

$$\exists f \in \mathbf{R}\ \forall n \in \mathbb{N}\ [\nu'(f(n)) = \Phi(\nu(n))].$$

(iii) Let $\Phi : \mathcal{X} \rightharpoonup \mathcal{X}$. Φ is a *(partial) endomorphism* of γ if $\Phi : \gamma \to \gamma$ ($\Phi : \gamma \rightharpoonup \gamma$).

As an immediate consequence of Eršov's fixedpoint theorem we get the following.

7.5.2. THEOREM. *Let $\gamma = \langle \mathcal{X}, \nu \rangle$ be a precomplete numeration. Let Φ be a partial endomorphism of γ. Then*

$$\exists a \in \mathcal{X} \ [\Phi(a)\downarrow \ \Rightarrow \ \Phi(a) = a].$$

(If Φ is an endomorphism then this yields

$$\exists a \in \mathcal{X} \ \ \Phi(a) = a.)$$

PROOF. Determine a lifting φ of Φ, and an 'Eršov-fixedpoint' n_0 of Φ. Now take $a = \nu(n_0)$. □

We can now try to characterize the endomorphisms of $\mathcal{PR}$ and $\mathcal{L}_\beta$. In the case of $\mathcal{PR}$, this is the Myhill-Shepherdson Theorem. We will recapitulate the argument because it is elegant; see also Barendregt and Longo (1983).

First the notion of recursive operator has to be introduced. The definition makes use of yet another numeration.

7.5.3. DEFINITION. (i) Recall that

$$\begin{aligned} W_e &= \mathrm{dom}(\{e\}) \\ &= \{x \in \mathbb{N} \mid \exists z \ T(e, x, z)\} \end{aligned}$$

where T is Kleene's T-predicate.

(ii) The numeration of *recursively enumerable sets* (notation $\mathcal{RE}$) is defined by

$$\mathcal{RE} = \langle \mathbf{RE}, W \rangle.$$

7.5.4. PROPOSITION. *$\mathcal{RE}$ is complete*

PROOF. The construction in the proof of Proposition 7.3.9 yields totalizations for $\mathcal{RE}$ as well. The special element is $\emptyset$. □

As a small digression, we can now give an interesting example of a morphism, given by the standard interpretation of λ-terms in the graph model $P\omega$ due to G. Plotkin and D.S. Scott.

7.5.5. FACT. Let $(E_n)_{n \in \mathbb{N}}$ be an effective enumeration of finite sets of natural numbers. Then $P\omega = \langle \wp(\mathbb{N}), \subseteq \rangle$ can be made into a λ-model via the operations application,

$$A \cdot B = \{m \in \mathbb{N} \mid \exists n \in \mathbb{N} \ [E_n \subseteq B \ \& \ \langle n, m \rangle \in A]\},$$

and abstraction: if $\Phi : P\omega \to P\omega$, then

$$\mathrm{graph}(\Phi) = \{\langle n, m \rangle \mid m \in \Phi(E_n)\}.$$

If Φ happens to be continuous one has

$$\mathrm{graph}(\Phi) \cdot B = \Phi(B).$$

7.5.6. DEFINITION. The interpretation of λ-terms in $P\omega$ is defined as follows.

$$\begin{aligned} [\![x]\!]_\rho &= \rho(x), \\ [\![MN]\!]_\rho &= [\![M]\!]_\rho \cdot [\![N]\!]_\rho, \\ [\![\lambda x.M]\!]_\rho &= \mathrm{graph}(\boldsymbol{\lambda} A.[\![M]\!]_{\rho(x:=A)}). \end{aligned}$$

The following shows that application and abstraction can be done effectively in $\mathcal{RE}$.

7.5.7. LEMMA. (i) *There is a recursive function* app *such that for all* e_1, e_2

$$W_{e_1} \cdot W_{e_2} = W_{\mathrm{app}(e_1,e_2)}.$$

(ii) *There is a recursive function* gr *such that for* $\mathcal{RE}$*-endomorphisms* Φ*, say with lifting* $\{e\}$

$$\mathrm{graph}(\Phi) = W_{\mathrm{gr}(e)}.$$

PROOF. (i) Easy.

(ii) First construct $\tilde{E} \in \mathbf{R}$ such that $E_n = W_{\tilde{E}(n)}$ for each n. Then

$$\mathrm{graph}(\Phi) = \{\langle n, m\rangle \mid m \in W_{\{e\}(\tilde{E}(n))}\}.$$

An index for this set can be found effectively from e. □

7.5.8. PROPOSITION. *Define* $\mathcal{I} : \mathbf{L}_\beta \to P\omega$ *by*

$$\mathcal{I}[M] = [\![M]\!].$$

Then $\mathcal{I}$ *is a morphism from* $\mathcal{L}_\beta$ *to* $\mathcal{RE}$.

PROOF. First note that the result of $\mathcal{I}$ is independent of the choice of representatives because $P\omega$ is a λ-model. Moreover note that environments ρ, mapping a finite set of λ-variables into $\mathbf{RE}$, can be effectively represented by finite sequences of the form

$$\langle\langle \sharp x_1, e_1\rangle, \ldots, \langle \sharp x_n, e_n\rangle\rangle,$$

with access function $\mathrm{lookup}(\rho, \sharp x)$ recursive. Now define, using the Recursion Theorem and the S-m-n-theorem, the function int, such that

$$\begin{aligned} \mathrm{int}(\sharp x, \rho) &= \mathrm{lookup}(\rho, \sharp x), \\ \mathrm{int}(\sharp MN, \rho) &= \mathrm{app}(\mathrm{int}(\sharp M, \rho), \mathrm{int}(\sharp N, \rho)), \\ \mathrm{int}(\sharp \lambda x.M, \rho) &= \mathrm{gr}(\tilde{\lambda} e.\mathrm{int}(\sharp M, \rho * \langle \sharp x, e\rangle)). \end{aligned}$$

Then we are done since $[\![M]\!] = W_{\mathrm{int}(\sharp M, \langle\rangle)}$. □

We continue the definition of recursive operators.

7.5.9. DEFINITION. (i) Let $\Phi : P\omega \to P\omega$. Then Φ is an *enumeration operator* (on $P\omega$) if

$$\Phi(X) = A \cdot X$$

for some $A \in \mathbf{RE}$. In that case we say that Φ is *defined by* A.

(ii) Let $\Phi : \mathbf{RE} \to \mathbf{RE}$. Φ is an *enumeration operator* (on $\mathbf{RE}$) if Φ is the restriction to $\mathbf{RE}$ of an enumeration operator on $P\omega$. (Note that $\mathbf{RE}$ is closed under application.)

Below we will show that the endomorphisms of $\mathcal{RE}$ are exactly the enumeration operators. The presentation is based on Barendregt and Longo (1983). The proof requires some topology.

7.5.10. DEFINITION. Let $\gamma = \langle \mathcal{X}, \nu \rangle$ be a numeration The *Eršov topology*on $\mathcal{X}$ induced by γ is obtained by taking as basic open sets the $\mathcal{A} \subseteq \mathcal{X}$ with $\nu^{-1}(\mathcal{A}) \in \mathbf{RE}$.

7.5.11. REMARK. Morphisms of numerations are continuous with respect to the Eršov topology.

Below the Eršov topology on $\mathbf{RE}$ (induced by $\mathcal{RE}$) is compared with the Scott topology.

7.5.12. DEFINITION. (i) The *Scott topology* on $\mathbf{RE}$ is the trace of the Scott topology on $\langle \wp(\mathbb{N}), \subseteq \rangle$; that is, the basic open sets are

$$\mathcal{B}_n = \{A \in \mathbf{RE} \mid e_n \subseteq A\},$$

and a set $\mathcal{A} \subseteq \mathbf{RE}$ is *Scott open* if

$$\mathcal{A} = \bigcup_{n \in I} \mathcal{B}_n \quad \text{for some } I \subseteq \mathbb{N}.$$

(ii) Let $\mathcal{A} \subseteq \mathbf{RE}$. Then $\mathcal{A}$ is *effectively Scott open* if

$$\mathcal{A} = \bigcup_{n \in I} \mathcal{B}_n \quad \text{for some } I \in \mathbf{RE}.$$

We abbreviate 'Eršov open', 'Scott open', etcetera to 'E-open', 'S-open', etcetera.

7.5.13. THEOREM (Rice-Shapiro). *Let* $\mathcal{A} \subseteq \mathbf{RE}$. *Then*

$$\mathcal{A} \text{ is basic E-open} \Leftrightarrow \mathcal{A} \text{ is effectively S-open.}$$

PROOF. See standard literature, e.g. Rogers (1967), Section 14.8. □

7.5.14. COROLLARY. (i) *Let* $\mathcal{A} \subseteq \mathbf{RE}$. *Then*

$$\mathcal{A} \text{ is E-open} \Leftrightarrow \mathcal{A} \text{ is S-open.}$$

(ii) *Let* $\Phi : \mathbf{RE} \to \mathbf{RE}$. *Then*

$$\Phi \text{ is E-continuous} \Leftrightarrow \Phi \text{ is S-continuous.}$$

This yields the desired result, which can be viewed as a Myhill-Shepherdson like theorem for **RE**.

7.5.15. THEOREM. *Let* $\Phi : \mathbf{RE} \to \mathbf{RE}$. *Then*

$$\Phi \text{ is an endomorphism of } \mathcal{RE} \Leftrightarrow \Phi \text{ is an enumeration operator.}$$

PROOF. ($\Rightarrow$) Suppose Φ is an endomorphism. Then Φ is E-continuous (Remark 7.5.11) and hence S-continuous by Corollary 7.5.14 (ii). One easily verifies that $\text{graph}(\Phi) \in \mathbf{RE}$. Hence Φ is an enumeration operator since

$$\Phi(X) = \text{graph}(\Phi) \cdot X.$$

($\Leftarrow$) All application functions

$$\lambda X.A \cdot X$$

are endomorphisms of $\mathcal{RE}$ by Lemma 7.5 7 (i). □

Let **PF** denote the class of unary partial functions.

7.5.16. DEFINITION. The *graph map* $\tau : \mathbf{PF} \hookrightarrow P\omega$ is defined by

$$\tau(\psi) = \{\langle x, y\rangle \mid \psi(x) \simeq y\}.$$

7.5.17. LEMMA. *The restriction of* τ *to* **PR** *is a morphism* $\tau : \mathcal{PR} \to \mathcal{RE}$.

PROOF. Easy. □

One might expect that the 'inverse' operation, i.e a partial $\tau^{-1} : \mathbf{RE} \rightharpoonup \mathbf{PR}$ is a partial morphism, but this is not the case. One has, however, the following Call $A \in \mathbf{RE}$ *graphical* if $A \in \tau(\mathbf{PR})$

7.5.18. LEMMA. *There exists a function* $\text{fun} \in \mathbf{R}$ *such that for all* $e \in \mathbb{N}$ *with* W_e *graphical*

$$\tau(\{\text{fun}(e)\}) = W_e.$$

PROOF. Define $\psi : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ by

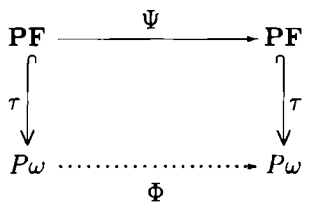
$$\psi(e, x) \simeq (\mu p[T(e, \langle x, (p)_0\rangle, (p)_1)])_0.$$

(Note that if W_e is graphical, then

$$\psi(e, x) \simeq \mu y[\langle x, y\rangle \in W_e].)$$

Then ψ is partial recursive. Let t be an index of ψ. Define $\mathrm{fun}(e) = S_1^1(t, e)$. □

7.5.19. DEFINITION. (i) A *partial recursive operator* is a map $\Psi : \mathbf{PF} \rightharpoonup \mathbf{PF}$ such that Ψ is defined by some enumeration operator Φ via τ:



i.e., if $\Psi = \tau^{-1} \circ \Phi \circ \tau$ with

$$\mathrm{dom}(\Psi) = \{\varphi \in \mathbf{PF} \mid \Phi(\tau(\varphi))\text{is graphical}\}.$$

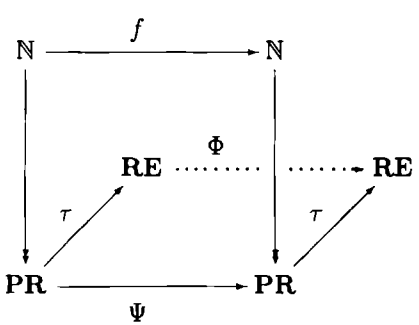
Ψ is a *recursive operator* if Ψ is total.

(ii) The notion of (partial) recursive operator is defined for **PR** as well by restricting operators to **PR** and **RE** respectively.

7.5.20 THEOREM (Myhill-Shepherdson). *Let* $\Psi : \mathbf{PR} \to \mathbf{PR}$. *Then*

$$\Psi \textit{ is an endomorphism of } \mathcal{PR} \Leftrightarrow \Psi \textit{ is a recursive operator.}$$

PROOF. ($\Rightarrow$) Suppose Ψ is an endomorphism with lifting f.

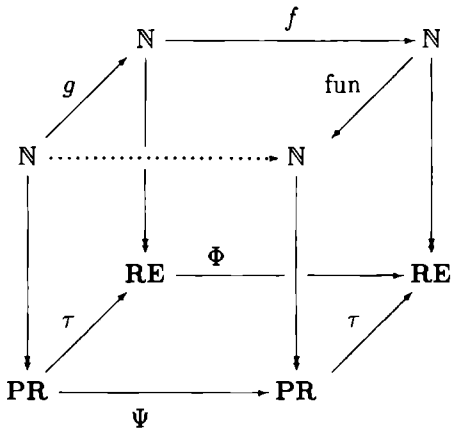


Define

$$F = \{\langle n, \langle p, q\rangle\rangle \mid e_n \text{ is graphical } \& \{f(\mathrm{fun}(n))\}(p) \simeq q\}.$$

Then $F \in \mathbf{RE}$, so F defines an enumeration operator Φ. Moreover Φ defines Ψ via τ. Hence Ψ is a recursive operator.

($\Leftarrow$) Suppose Ψ is defined by an enumeration operator Φ. By Theorem 7.5.15 Φ has a lifting f. By Lemma 7.5.17, the graph map τ has a lifting, say g. Using the function fun, constructed in Lemma 7.5.18 we can erect the following diagram.



Now fun $\circ f \circ g$ lifts Ψ. (Note that $\Phi(\tau(\psi))$ is graphical for each $\psi \in \mathbf{PR}$ since Ψ is total.) □

7.5.21. COROLLARY. *Each recursive operator has a fixedpoint in* **PR**.

PROOF. By theorems 7.5.2 and 7.5.20. □

This is almost the first recursion theorem; in the next section we will discuss the minimality of fixed points.

No we study endomorphisms of $\mathcal{L}_\beta$. We can only give a partial characterization.

7.5.22. PROPOSITION. *All right-application functions*

$$\text{“}\boldsymbol{\lambda}[M].[FM]\text{”}$$

are endomorphisms of $\mathcal{L}_\beta$.

PROOF. Easy. □

This gives us (a restricted version of) the first λ-calculus fixedpoint theorem.

7.5.23. COROLLARY. $\forall F \in \Lambda^\circ \ \exists X \in \Lambda^\circ \ FX =_\beta X$.

Contrary to the situation in $\mathcal{RE}$, not all endomorphisms are right-applications. Below, we give a counterexample suggested by R. Statman. For $P, Q \in \Lambda$, $P \circ Q$ stands for $\lambda x.P(Qx)$.

7 5 24 DEFINITION Suppose $F, G \in \Lambda^\circ$ such that F is the left inverse of G, i e $F \circ G =_\beta \mathbf{I}$ This gives rise to a so called *inner λ-model*, by translating lambda terms M to $(\!|M|\!)^{FG}$, defined inductively as follows

$$\begin{aligned} (\!|x|\!)^{FG} &\equiv x, \\ (\!|MN|\!)^{FG} &\equiv F(\!|M|\!)^{FG}(\!|N|\!)^{FG}, \\ (\!|\lambda x\, M|\!)^{FG} &\equiv G(\lambda x\, (\!|M|\!)^{F,G}) \end{aligned}$$

7 5 25 LEMMA *For all* $M, N \in \Lambda$

$$M =_\beta N \;\Rightarrow\; (\!|M|\!)^{FG} =_\beta (\!|N|\!)^{F,G}$$

PROOF By induction on the generation of $=_\beta$ Let us check the basic case of $\rightarrow_\beta$

$$\begin{aligned} (\!|(\lambda x\, P)Q|\!)^{FG} &\equiv F(G(\lambda x\, (\!|P|\!)^{FG}))(\!|Q|\!)^{FG} \\ &=_\beta (\lambda x\, (\!|P|\!)^{F,G})(\!|Q|\!)^{FG} \\ &=_\beta (\!|P|\!)^{F,G}[x := (\!|Q|\!)^{FG}] \end{aligned}$$

By an induction on P one easily shows that the latter term is equal to $(\!|P[x := Q]|\!)^{FG}$ and we are done □

7 5 26 DEFINITION The map $[\![\ ]\!]\ \mathbf{L}_\beta \rightarrow \mathbf{L}_\beta$ is defined by

$$[\![[M]]\!] = [(\!|M|\!)^{F,G}]$$

7 5 27 PROPOSITION $[\![\ ]\!]$ *is an endomorphism of* $\mathcal{L}_\beta$

PROOF By Lemma 7 5 25 and the fact that $(\!|\ |\!)^{FG}$ is effective □

Now set

$$\begin{aligned} F &\equiv \lambda p\, p\mathbf{I}, \\ G &\equiv \lambda pq\, qp \end{aligned}$$

Then indeed $F \circ G =_\beta \mathbf{I}$

7 5 28 PROPOSITION *There is no* $H \in \Lambda^\circ$ *such that for each* $M \in \Lambda^\circ$

$$HM =_\beta (\!|M|\!)^{FG}$$

PROOF Consider $(\!|\mathbf{\Omega}|\!)^{FG} \equiv F(G(\lambda x\, Fxx))(G(\lambda x\, Fxx))$ By analyzing the reduction possibilities of $(\!|\mathbf{\Omega}|\!)^{FG}$ (e g , using the cofinal Gross-Knuth reduction strategy, see Barendregt (1984), Section 13 2) one can conclude that $\mathbf{\Omega}$ does not occur as a subterm in any reduct of $(\!|\mathbf{\Omega}|\!)^{F,G}$ Now suppose H satisfies the condition Then $H\mathbf{\Omega}$ and $(\!|\mathbf{\Omega}|\!)^{FG}$ have a common reduct without $\mathbf{\Omega}$ Since $\mathbf{\Omega}$ is of order 0 (it behaves like a variable) one has that even Hx reduces to a reduct of $(\!|\mathbf{\Omega}|\!)^{FG}$, so $Hx =_\beta (\!|\mathbf{\Omega}|\!)^{FG}$ $(\neq_\beta x)$, contradiction □

7.6. Effective Versions of the Recursion Theorems

The effective version of Eršov's fixed point theorem (Proposition 7.3 12) yields an effective formulation of the Second Recursion Theorem: there is a function $g \in \mathbf{R}$ such that for all e with $\{e\}$ total

$$\{\{e\}(g(e))\} = \{g(e)\}.$$

In the sequel we obtain an effective version of the First Recursion Theorem, which can be formulated as follows.

FIRST RECURSION THEOREM. *Each recursive operator has a least fixed point in* **PR**.

Note that the effective version of the second recursion theorem yields a fixed point of each recursive operator via an index of a lifting, but the 'Eršov construction' does not necessarily give an index of the *least* fixed point. The following theorem will be proved.

THEOREM. *There exists* $g \in \mathbf{R}$ *such that for each recursive operator* Ψ

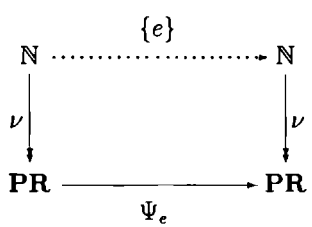
$$\forall e \in \mathbb{N}\ [\{e\} \textit{ lifts } \Psi \Rightarrow \{g(e)\} \textit{ is the least fixed point of } \Psi].$$

For convenience the following denotation is introduced. Note that all liftings of endomorphisms are '$\sim_\gamma$-faithful' recursive functions, and vice versa.

7.6.1. DEFINITION. (i) The set FF of $\sim_{\mathcal{PR}}$*-faithful indices* is defined as follows.

$$\begin{aligned} \mathrm{FF} &= \{e \in \mathbb{N} \mid \{e\} \text{ is } \sim_{\mathcal{PR}}\text{-faithful}\} \\ &= \{e \in \mathbb{N} \mid \{e\} \text{ is total } \& \ \forall m, n \in \mathbb{N}\ [m \sim_{\mathcal{PR}} n \Rightarrow \{e\}(m) \sim_{\mathcal{PR}} \{e\}(n)]\}. \end{aligned}$$

(ii) Let $e \in \mathrm{FF}$. Then Ψ_e is the (unique) recursive operator of which $\{e\}$ is a lifting.



Now the statement in the theorem can be reformulated.

7.6.2. THEOREM (Effective first recursion theorem)

$$\exists g \in \mathbf{R}\ \forall e \in \mathrm{FF}\ [\{g(e)\} \textit{ is the least fixed point of } \Psi_e].$$

The proof occupies 7.6 3–7.6.6.

Since each enumeration operator is defined by an element of **RE**, each set $A \in \mathbf{RE}$ gives rise to a (partial) recursive operator Ψ_A with

$$\Psi_A(\psi) = \tau^{-1}(A \cdot \tau(\varphi)).$$

$$\begin{array}{ccc} \mathbf{PR} & \xrightarrow{\Psi} & \mathbf{PR} \\ {\scriptstyle\tau}\downarrow & & \downarrow{\scriptstyle\tau} \\ \mathbf{RE} & \xrightarrow[\lambda X.A \cdot X]{} & \mathbf{RE} \end{array}$$

NOTATION. Let $e \in \mathbb{N}$. Then $\tilde{\Psi}_e$ is the (partial) recursive operator defined by W_e, that is

$$\tilde{\Psi}_e = \Psi_{W_e}.$$

Recall that every continuous function on $P\omega$ has a least fixed point.

7.6.3. DEFINITION. Let Y be the least-fixed-point operator on $P\omega$. That is, for all $A \in P\omega$ $Y(A)$ is the least fixed point of the function $\lambda X.A \cdot X$.

7.6.4. LEMMA. *The restriction of Y to* **RE** *is an endomorphism of $\mathcal{RE}$: there is a lifting k such that the following diagram commutes.*

$$\begin{array}{ccc} \mathbb{N} & \xrightarrow{k} & \mathbb{N} \\ {\scriptstyle W}\downarrow & & \downarrow{\scriptstyle W} \\ \mathbf{RE} & \xrightarrow[Y]{} & \mathbf{RE} \end{array}$$

PROOF. Y is defined by $[\![\mathbf{Y}]\!]^{P\omega}$, where $\mathbf{Y}$ is the λ-calculus fixed point combinator

$$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)),$$

see Barendregt (1984), Theorem 19.3.4. Note that $[\![\mathbf{Y}]\!]^{P\omega} \in \mathbf{RE}$ since $\mathbf{Y}$ is closed, by Proposition 7.5.8. Therefore Y is an enumeration operator and hence by Theorem 7.5.15 an endomorphism of $\mathcal{RE}$. □

7.6.5. LEMMA. $\exists w \in \mathbf{R}\ \forall e \in \mathrm{FF}\ \ \Psi_e = \tilde{\Psi}_{w(e)}$.

PROOF. This is the effective version of the construction of F in the proof of Theorem 7.5.20. □

7 6 6 LEMMA *If* $\Phi = \lambda X\, A\ \ X$ *defines a total recursive operator via* τ, *then* $Y(A)$ *is graphical*

PROOF If $X \in \mathbf{RE}$ is graphical, then $A\ \ X$ is graphical too since Φ defines a total operator Now note that

$$Y(A) = \bigcup_{n \in \mathbf{N}} A^n(\emptyset),$$

and that $\emptyset$ is graphical □

PROOF OF THE THEOREM Take $g = \mathrm{fun} \circ k \circ w$ Then g satisfies the requirements by the lemmas 7 5 18, 7 6 4 and 7 6 5 Note that Lemma 7 6 6 guarantees that $W_{k(w(e))}$ is graphical for each $e \in \mathrm{FF}$ □

Bibliography

Abramsky, S., D.M. Gabbay and T.S.E. Maibaum (eds.) (1992). *Handbook of Logic in Computer Science*, Vol. II, Oxford University Press.

Achten, P.M., J.H.G. van Groningen and M.J. Plasmeijer (1993). High level specification of I/O in functional languages, *Proceedings of the International Workshop on Functional Languages*, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Ariola, Z.M. and J.W. Klop (1993). Equational term graph rewriting, Draft.

van Bakel, S.J., J.E.W. Smetsers and S. Brock (1992). Partial type assignment in left-linear term rewriting systems, *in:* J.C. Raoult (ed.), *Proceedings of the 17th Colloquium on Trees and Algebra in Programming (CAAP'92)*, Rennes, France, Lecture Notes in Computer Science 581, Springer-Verlag, Berlin, pp. 300–322.

de Bakker, J.W., A.J. Nijman and P.C. Treleaven (eds.) (1987). *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE)* II, Eindhoven, The Netherlands, Lecture Notes in Computer Science 259, Springer-Verlag, Berlin.

Barendregt, H.P. (1984). *The Lambda Calculus: Its Syntax and Semantics*, second, revised edition, North-Holland, Amsterdam.

Barendregt, H.P. (1991). Self-interpretation in λ-calculus, *Journal of Functional Programming* **1**, pp. 229–234.

Barendregt, H.P. (1992) Lambda calculi with types, *in:* Abramsky et al. (1992).

Barendregt, H.P. and G. Longo (1983). Recursion theoretic operators and morphisms on numbered sets, *Fundamenta Mathematicae* **CXIX**, pp. 49–62.

Barendregt, H.P., M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep (1987a). Term graph reduction, *in:* de Bakker et al. (1987), pp. 141–158.

Barendregt, H.P., M.C.J.D. van Eekelen, J.R.W. Glauert, J.R Kennaway, M.J. Plasmeijer and M.R. Sleep (1987b). Towards an intermediate language based on graph rewriting, *in:* de Bakker et al. (1987), pp. 159–175.

Barendsen, E. (1990). Semantics of simply typed lambda calculus, Typescript, University of Nijmegen.

Barendsen, E. (1991). An unsolvable numeral system in lambda calculus, *Journal of Functional Programming* **1**, pp. 367–372.

Barendsen, E. and M.A. Bezem (1991). Bar recursion versus polymorphism (extended abstract), *Technical Report 69*, Logic Group, Utrecht Research Institute for Philosophy, Utrecht University.

Barendsen, E. and M.A. Bezem (1992). Bar recursion versus polymorphism, *Technical Report 81*, Logic Group, Utrecht Research Institute for Philosophy, Utrecht University.

Barendsen, E. and J.E.W. Smetsers (1992). Graph rewriting and copying, *Technical Report 92-20*, Computing Science Institute, University of Nijmegen.

Barendsen, E. and J.E.W. Smetsers (1993a). Conventional and uniqueness typing in graph rewrite systems, *Technical Report CSI-R9328*, Computing Science Institute, University of Nijmegen.

Barendsen, E. and J.E.W. Smetsers (1993b). Conventional and uniqueness typing in graph rewrite systems (extended abstract), *in:* R.K. Shyamasundar (ed.), *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, Lecture Notes in Computer Science 761, Springer-Verlag, Berlin, pp. 41–51.

Barendsen, E. and J.E.W. Smetsers (1994). Extending graph rewriting with copying, *in:* Schneider and Ehrig (1994), pp. 51–70.

Bezem, M.A. (1985). Strongly majorizable functionals of finite type: A model for bar recursion containing discontinuous functionals, *Journal of Symbolic Logic* **50**, pp. 652–660.

Bezem, M.A. (1986). *Bar Recursion and Functionals of Finite Type*, Dissertation, Utrecht University.

Breazu-Tannen, V. and Th Coquand (1988). Extensional models for polymorphism, *Theoretical Computer Science* **59**, pp. 85–114.

Breazu-Tannen, V. and A. Meyer (1987). Polymorphism is conservative over simple types, *Proceedings of the 2nd Annual Symposium on Logic in Computer Science*, Ithaca, New York, IEEE Computer Society Press, pp. 7–17.

Bruce, K.B., A. Meyer and J.C. Mitchell (1990). The semantics of second-order lambda calculus, *Information and Computation* **85**, pp. 76–134.

Brus, T., M.C.J.D. van Eekelen, M.O. van Leer and M.J. Plasmeijer (1987). Clean: A language for functional graph rewriting, *Proceedings of the Conference on Functional Languages and Computer Architectures (FPCA)*, Portland, Oregon, Lecture Notes in Computer Science 274, Springer-Verlag, Berlin, pp. 364–384.

Church, A. (1936). An unsolvable problem of elementary number theory, *American Journal of Mathematics* **58**, pp. 354–363.

Church, A. (1940). A formulation of the simple theory of types, *Journal of Symbolic Logic* **5**, pp. 56–68.

van Draanen, J.P. (1989). *Lambda-definability and partial recursiveness in higher types*, Master's thesis, Department of Computing Science, University of Nijmegen.

Ehrig, H., M. Nagl, G. Rozenberg and A. Rosenfeld (eds.) (1987). *Proceedings of the 3rd International Workshop on Graph-Grammars and their Application to Computer Science*, Warrenton, Virginia, Lecture Notes in Computer Science 291, Springer-Verlag, Berlin.

Eršov, J.L. (1973). Theorie der Numerierungen, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **19**, pp. 289–388.

Fortune, S., D. Leivant and M. O'Donnell (1983). The expressiveness of simple and second-order type structures, *Journal of the Association for Computing Machinery* **30**, pp. 151–185.

Friedman, H. (1975). Equality between functionals, *in:* R. Parikh (ed.), *Logic Colloquium: Symposium on Logic held at Boston*, Lecture Notes in Mathematics 453, Springer-Verlag, Berlin, pp 22–37.

Geuvers, J.H. (1993). *Logics and Type Systems*, Dissertation, University of Nijmegen.

Girard, J.-Y. (1972) *Interprétation fonctionelle et élimination des coupures dans l'arithétique d'ordre supérieur*, Dissertation, Université Paris VII.

Girard, J.-Y., Y. Lafont and P. Taylor (1989). *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press.

Gödel, K. (1958). Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes, *Dialectica* **12**, pp. 280–287.

Goodstein, R.L. (1941). Function theory in an axiom-free equation calculus, *Proceedings of the London Mathematical Society* **48**, pp. 401–434.

Guzmán, J C. and P. Hudak (1990). Single-threaded polymorphic lambda calculus, *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, Philadelphia, IEEE Computer Society Press, pp. 333–343.

Heyting, A. (ed.) (1959). *Constructivity in Mathematics*, North-Holland, Amsterdam.

Hindley, J.R. (1964). *The Church-Rosser Property and a Result in Combinatory Logic*, Dissertation, University of Newcastle-upon-Tyne.

Hinman, P.G. (1978). *Recursion-theoretic Hierarchies*, North-Holland, Amsterdam.

Howard, W.A. (1968). Functional interpretation of bar induction by bar recursion, *Compositio Mathematica* **20**, pp. 107–124.

Howard, W.A. (1973). Hereditarily majorizable functionals of finite type, *in:* Troelstra (1973), pp 455–459.

Jacobs, B.P.F. (1993). Personal communication.

Jäger, G. and Th. Strahm (1994). Totality in applicative theories, To appear in Annals of Pure and Applied Logic.

Kechris, A.S. and Y.N. Moschovakis (1977). Recursion in higher types, *in:* J. Barwise (ed.), *Handbook of Mathematical Logic*, North-Holland, Amsterdam, pp. 681–737.

Kleene, S.C. (1959a). Countable functionals, *in:* Heyting (1959), pp. 81–100.

Kleene, S.C. (1959b). Recursive functionals and quantifiers of finite type I, *Transactions of the American Mathematical Society* **91**, pp. 1–52

Kleene, S.C. (1962). Lambda-definable functionals of finite types, *Fundamenta Mathematicae* **L**, pp. 281–303.

Klop, J.W. (1992) Term rewrite systems, *in:* Abramsky et al. (1992)

Kreisel, G (1959). Interpretation of analysis by means of constructive functionals of finite type, *in:* Heyting (1959), pp. 101–128.

Kreisel, G., D. Lacombe and J.R. Schoenfield (1959) Partial recursive functionals and effective operations, *in:* Heyting (1959), pp. 290–297.

Luckhardt, H. (1975). The real elements in a consistency proof for simple type theory, *in:* J. Diller and G.H. Müller (eds.), *Proof Theory Symposium Kiel 1974*, Lecture Notes in Mathematics 500, Springer-Verlag, Berlin, pp. 233–256.

Myhill, J. and J.C. Shepherdson (1955). Effective operations on partial recursive functions, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **1**, pp. 310–317.

Nöcker, E.G.J.M.H., J.E.W. Smetsers, M.C.J.D. van Eekelen and M.J. Plasmeijer (1991). Concurrent Clean, *in:* E.H.L. Aarts, J. van Leeuwen and M. Rem (eds.), *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE)* II, Eindhoven, The Netherlands, Lecture Notes in Computer Science 505, Springer-Verlag, Berlin, pp. 202–219.

Plasmeijer, M.J. and M.C.J.D. van Eekelen (1993). *Functional Programming and Parallel Graph Rewriting*, Addison-Wesley.

Poll, E. (1994). *A Programming Logic Based on Type Theory*, Dissertation, Eindhoven University of Technology.

Reynolds, J.C. (1974). Towards a theory of type structure, *Colloque sur la Démonstration*, Paris, Lecture Notes in Computer Science 19, Springer-Verlag, Berlin, pp. 408–425.

Reynolds, J.C. (1984). Polymorphism is not set-theoretic, *in:* G. Kahn, D.B. McQueen and G. Plotkin (eds.), *Semantics of Data Types: international symposium*, Sophia-Antipolis, France, Lecture Notes in Computer Science 173, Springer-Verlag, Berlin, pp. 145–156.

Robinson, J.A. (1965). A machine-oriented logic based on the resolution principle, *Journal of the Association for Computing Machinery* **12**, pp. 23–41.

Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*, McGraw Hill, New York.

Rosen, B.K. (1973). Tree manipulation systems and Church-Rosser theorems, *Journal of the Association for Computing Machinery* **20**, pp. 160–187.

Sastry, W., A.V.S. Clinger and Z. Ariola (1993). Order-of-evaluation analysis for destructive updates in strict functional languages with flat aggregates, *Proceedings of the Conference on Functional Languages and Computer Architectures (FPCA)*, Copenhagen, Denmark, ACM Press, pp. 266–276.

Schneider, H.J. and H. Ehrig (eds.) (1994). *Graph Transformations in Computer Science, International Workshop*, Dagstuhl Castle, Germany, Lecture Notes in Computer Science 776, Springer-Verlag, Berlin.

Smetsers, J.E.W., E. Barendsen, M.C.J.D. van Eekelen and M.J. Plasmeijer (1994). Guaranteeing safe destructive updates through a type system with uniqueness information for graphs, *in:* Schneider and Ehrig (1994), pp. 358–379.

Smullyan, R M. (1985). *To Mock a Mockingbird and Other Logic Puzzles: Including an Amazing Adventure in Combinatory Logic*, Knopf, New York.

Spector, C. (1962). Provably recursive functionals of analysis: A consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics, *in:* J.C.E. Dekker (ed.), *Recursive Function Theory*, Proceedings of Symposia in Pure Mathematics V, American Mathematical Society, Providence, pp. 1–27.

Tait, W. (1967). Intensional interpretations of functionals of finite type I, *Journal of Symbolic Logic* **32**, pp. 198–212.

Troelstra, A.S. (1992). *Lectures on Linear Logic*, CSLI Lecture Notes 29, CSLI, Stanford.

Troelstra, A.S. (ed.) (1973). *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, Lecture Notes in Mathematics 344, Springer-Verlag, Berlin.

Turing, A.M. (1936/7). On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* **42**, pp. 230–265.

Turner, D.A. (1985). Miranda: A non-strict functional language with polymorphic types, *in:* J.P. Jouannaud (ed), *Proceedings of the Conference on Functional Languages and Computer Architectures (FPCA)*, Nancy, France, Lecture Notes in Computer Science 201, Springer-Verlag, Berlin, pp. 1–16.

Visser, A. (1980). Numerations, λ-calculus and arithmetic, *in:* J.R. Hindley and J.P. Seldin (eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, Academic Press, New York, pp. 259–284

Wadler, P. (1990). Linear types can change the world!, *Proceedings of the Working Conference on Programming Concepts and Methods*, Israel, North-Holland, Amsterdam, pp. 385–407.

Wadsworth, C.P. (1971). *Semantics and Pragmatics of the Lambda Calculus*, Dissertation, Oxford University.

Wand, M. (1987). A simple algorithm and proof for type inference, *Fundamenta Informaticae* **X**, pp. 115–122.

Index

admissible, 112
algebraic type system, 91
application rule, 88

bar recursion, 59
bar recursor, 60

Church numeral, 15
Church-Rosser, 17
classification, 107, 108, 110
coercion, 104, 115
  — consistent, 120
  — safe, 120
compatible, 86
complete, 162
contractum root, 87
critical path combination, 109
Curried version, 96
Curry closed, 89
Curry variant, 88
cycle, 84

domain, 38

endomorphism, 166
enumeration operator, 169
erase-type map, 40
Eršov topology, 169
extension, 87
extensionality, 14, 32

first-order fragment, 46
fixed point theorem, 163
  first λ-calculus —, 159, 172
  Godel's —, 160
  second λ-calculus —, 164
formulas, 31

functional reduction strategy, 110

garbage collection, 87
graph, 83

HEO, 45
HEO2, 44
Hindley-Rosen Lemma, 19

interpretation, 34, 39, 41, 42

joining node, 109

Kleene closed, 56

λ-closed, 47
$\boldsymbol{\lambda}\Box$-structure, 34
lifting, 166

marking
  — for graph, 109
  — for rewrite rule, 118
  saturated —, 131
match, 84, 86
model, 35, 43
  inner λ- —, 173
morphism, 166

numeration, 161
  λ calculus —, 162
  — of p r functions, 163
  — of r e sets, 167
  Peano arithmetic —, 165

occurrence increasing, 120
ω-model, 65
ω-structure, 14
oracle, 14

partial applicative structure, 37
partial equivalence relation, 37
partial Kleene recursive, 56
path, 83
pattern symbol, 86
per structure, 38
  — for $\lambda\Box$, 40
  — over $\mathfrak{M}$, 46
plain polymorphic extension, 33
precomplete, 161
primary, 107
primitive recursor, 32
Principal Type Theorem, 94

quantifier-free arithmetic, 61

realizable, 66
realization equivalent, 67
recursion theorem
  first —, 174
    effective —, 174
  second —, 165
recursive operator, 171
redex, 86
redirection, 87
representation
  R- —, 15
  standard —, 15
rewrite rule, 84, 85

Scott topology, 169
secondary, 107
sign, 114
sign classification lattice, 114
simple, 107
specification, 66
stack, 138
  — safe, 139
  — suspect, 139
subgraph, 84
subject reduction, 97, 100, 154
substitution, 90, 116

term graph rewrite system, 88
terms
  $\lambda\Box$- —, 30
  $\lambda\mathfrak{M}$- —, 14
totalize, 161
tree, 84
type environment, 92, 117
  algebraic —, 95, 122
  function —, 96, 124
type structure, 13
  full —, 14
types
  first-order —, 13, 29
  graph —, 90
  polymorphic —, 29
  second-order —, 29
  simple —, 29
  symbol —, 90
  uniqueness —, 113
typing
  — for graph, 92, 117
  — for rewrite rule, 95, 118

underlining, 22
unifier, 91
unique, 130
uniqueness
  — attributes, 111
  — propagation, 105, 112, 120
  — removal map, 132
  — safe, 120
  — types, 113
  — typing
    — for graph, 117
    — for rewrite rule, 118
    plain —, 118

Samenvatting

Dit proefschrift gaat over *typering*

Een *typeringssysteem* is een formalisme om types toe te kennen aan objecten Een typeringsuitspraak in zo'n systeem is van de vorm $a \quad \tau$, waarin a een object is en τ een type

Deze typeringsuitspraken zijn op twee manieren te interpreteren De eerste vat a op als een programma of algoritme, en τ als een specificatie De geldigheid van $a \quad \tau$ betekent dan dat het programma aan de specificatie voldoet

De tweede interpretatie ziet τ als een formule in een of ander logisch systeem, en a als een bewijs Hier drukt $a \quad \tau$ uit dat a een bewijs is van τ

In dit proefschrift staat het eerste, berekeningstechnische, gezichtspunt centraal Voor het onderzoeken van theoretische eigenschappen van programmeertalen zijn de objecten steeds elementen van een zeker *berekeningsmodel* Een berekeningsmodel bestaat uit een verzameling objecten en rekenregels die de essentiele aspecten van het gedrag van een te bestuderen computertaal bevatten Zo'n model is te zien als een gestileerde computertaal in beginsel even krachtig als het origineel, echter niet bedoeld om serieus in te programmeren

Karakteristieke punten van onderzoek zijn dan de uitdrukkingskracht van het typeringssysteem ('Welke operaties zijn uit te drukken in een typeerbaar programma?') en de interactie van typering en de rekenregels van het model ('Hoe beinvloedt het uitvoeren van een programma de typering?')

In dit werk komen twee berekeningsmodellen voor functionele programmeertalen aan de orde de *lambda-calculus* en *graafherschrijfsystemen*

De λ-calculus is het eenvoudigste van de twee In deel I bestuderen we de uitdrukkingskracht van twee uitbreidingen van K Godel's systeem $\boldsymbol{\lambda T}$ van primitief-recursieve functionalen ten eerste het systeem $\boldsymbol{\lambda TB}$ van bar-recursieve functionalen (verkregen door toevoeging van een nieuw rekenprincipe) van C Spector, en ten tweede de polymorfe λ-calculus $\boldsymbol{\lambda 2T}$ (waarin de types worden uitgebreid met polymorfie) van J Y Girard

Ondanks het verschil tussen deze twee calculi zijn ze even krachtig ten aanzien van definieerbaarheid van numerieke functies, dat wil zeggen operaties met natuurlijke getallen als in- en uitvoerobjecten Als eerste aanzet tot een nadere vergelijking van $\boldsymbol{\lambda TB}$ en $\boldsymbol{\lambda 2T}$ richten we ons op definieerbaarheid van *functionalen* operaties met functies als invoerobjecten, en numerieke uitvoer Naast een

bewijs- en modeltheoretische karakterisering van het begrip 'defineerbaarheid' voor functionalen levert dit een operatie op die defineerbaar is in **λTB** maar niet in **λ2T** De modeltheoretische techniek om dit aan te tonen is interessant op zich zelf

Door een modelconstructie gebaseerd op partiele equivalentierelaties toe te passen op de uitbreiding van (ongetypeerde) λ-calculus met hogere-orde functionalen kan een model van **λ2T** verkregen worden waarin de operatie in kwestie niet aanwezig is De constructie kan algemeen gebruikt worden om een ruime klasse van modellen van simpel getypeerde λ-calculus uit te breiden naar modellen van polymorfe λ-calculus

Graafherschrijfsystemen vormen een berekeningsmodel dat dichter bij zowel de vorm als de implementatie van functionele programmeertalen staat Net als in de verwante termherschrijfsystemen is hierin de definitie van functies gescheiden van hun toepassing Operaties kunnen worden gedefinieerd middels zogenaamde herschrijfregels De objecten zijn grafen meervoudig gebruik van een deelobject komt tot uiting door het bestaan van meer dan één referentie ('pijl') naar dat deelobject Zo zijn ook cyclische structuren mogelijk

In deel II wordt een graaftheoretische versie van een klassiek typeringssysteem in de stijl van H B Curry beschreven Zo'n systeem, met een beperkte polymorfie en algebraisch gedefinieerde datatypes, is aanwezig in de meeste functionele programmeertalen We bewijzen dat in dit typeringssysteem elke typeerbare graaf een standaardtype heeft, waaruit elk ander type door consistente substitutie verkregen kan worden

We breiden dit typeringssysteem uit met zogenaamde *uniciteitsinformatie* Daarbij is niet alleen de vorm van een (deel)object van belang, maar ook het aantal verwijzingen ernaar Het aldus verkregen systeem is verwant aan lineaire logica Uniciteitstypering kan gebruikt worden om functionele programmeertalen uit te breiden met primitieven voor interactie met bijvoorbeeld een bedrijfssysteem voor bestandsbeheer, zonder het functionele karakter geweld aan te doen In de programmeertaal *Clean* is uniciteitstypering inmiddels ingebouwd en succesvol toegepast, onder andere voor genoemde primitieven

Het typeringssysteem bevat een annotatiemechanisme voor conventionele types en een vorm van subtypering We beschrijven een verfijnd mechanisme om referenties te tellen, waarbij in zekere mate rekening wordt gehouden met de structuur van de graaf na toekomstige herschrijfstappen Typering in dit systeem blijft behouden tijdens herschrijven Het bewijs van dit resultaat vergt onder meer een complexe analyse van referentiepaden in grafen

Deel III bevat tenslotte enkele bespiegelingen over berekenbare operaties op λ-termen en recursieve functionalen Analyse in J L Eršovs raamwerk van opsommingen levert een karakterisering van twee soorten dekpuntresultaten

Curriculum Vitae

Erik Barendsen werd op 10 juni 1966 in Zutphen geboren. In die stad bezocht hij vanaf 1978 het Stedelijk Lyceum. Zijn middelbare-schooltijd werd in 1984 afgesloten met het diploma OVWO.

Van 1984 tot 1989 studeerde hij Wiskunde en Informatica aan de Katholieke Universiteit Nijmegen. Met name door colleges van H.P. Barendregt en W.H.M. Veldman werd zijn belangstelling voor het grondslagenonderzoek gewekt. Hij studeerde in 1989 cum laude af in genoemde vrije studierichting, met specialisatie Grondslagen van de Informatica.

Van 1989 tot 1993 was hij als onderzoeker in opleiding (in dienst van NWO) werkzaam in de afdeling Grondslagen van de vakgroep Informatica aan de Katholieke Universiteit Nijmegen. Naast het onderzoek in het kader van het SION-project 'Syntaxis en semantiek van termen en types' leverde hij een bijdrage aan het onderwijs in de fundamentele informatica. Sinds 1993 is hij als universitair docent in (tijdelijke) dienst van de KUN.